Reliability-Focused Scheduling with (m, k)-firm Deadlines over Wireless Channels – A Reinforcement-Learning Approach

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master or Science
by

Yakir Matusovsky

University of Canterbury
2016

This work is dedicated to my beloved wife Maya who has encouraged me to accept this challenge and to my son Alex and my daughter Emma without whose participation this process would have been much less joyful.

Acknowledgments

I would like to thank my supervisors Professor Andreas Willig and Doctor Adriel Kind for their comments, guidance and tireless work.

Abstract

In wireless radio applications, the quality of an underlying wireless channel is important, however, we know of a few applications that can tolerate some losses. As an example, real-time applications like streaming voice or video do permit packet loss and still retain a bearable service. With respect to quality of service requirement, we embrace one concise method to distinguish between the allowed and forbidden loss patterns. This method is known as the (m, k)-firm deadlines; at least m out of k consecutive packets have to be successfully delivered to their destination. We consider a point to multi-point network with a known population of wireless communication terminals and with one base station periodically polling all terminals. Given limited network resources, a recovery from data losses in such arrangement might be very challenging under high error rates and with large number of nodes. In this thesis, we consider policies that improve quality of multiple periodic streams by retransmission of failed packets. The base scheduler decides which streams to serve with respect to the primary goal of minimizing violation of the stream's deadline. We introduce an algorithm from Reinforcement Learning theory and compare its performance to a few baseline scheduling policies with static channels and channels with time-varying characteristics. We found that although the Learning scheme introduces good performance it doesn't outperform a baseline technique which is based on immediate slot allocation decisions with respect to packet error rate of each stream.

Glossary

CSMA Carrier Sense Multiple Access – A shared Medium access protocol in which a user senses other traffic before trying to transmit.

EDF Earliest deadline first – A scheduling algorithm in which on the next scheduled event, e.g. of running a process, such will be selected by the highest proximity to the execution deadline.

LTE Long Term Evolution – A standard for high-speed wireless communications in mobile networks.

MDP Markov Decision Process – A discrete-time stochastic process in control theory. The process resides in some state s. The decision maker may choose any action a available from the state. The process then will randomly move to another state s' and result in reward $r(s, s')$.

PROFIBUS Process Field Bus, a standard for fieldbus communication in automation technology, 1989. Openly published as part of IEC 61158.

SCADA Supervisory Control And Data Acquisition – A system for remote monitoring and control that operates over communication channels; typical applications are in power, water, nuclear and other utility industries.

TDMA Time division multiple access is a channel access method for networks that use shared mediums. TDMA allows many users to work on a common frequency channel by dividing the signal into different time slots.

Acronyms

ACK Acknowledgment.

ARQ Automatic Repeat Request.

DBP Distance Based Priority.

DTV Distance to Violation.

eACK Explicit Acknowledgment.

FEC Forward Error Correction.

GLIE Greedy in the Limit with Infinite Exploration.

GPI Generalized Policy Iteration.

HRT Hard Real Time.

ICN Industrial Communication Networks.

LAN Local Area Network.

MAC Medium Access Control Layer.

MARL Multi-Agent Reinforcement Learning.

MC Monte Carlo.

MDP Markov Decision Process.

MPEG Moving Picture Experts Group.

NACK Negative Acknowledgment.

NE Nash Equilibrium.

PER Packet Error Rate.

QoS Quality of Service.

RDT Redundant Data Transmission.

RL Reinforcement Learning.

RR Round Robin.

RTT Round Trip Time.

SRT Soft Real Time.

TCP Transport Control Protocol.

TD Temporal Difference.

WHRT Weakly-Hard Real Time.

WSN Wireless Sensor Networks.

Table of Contents

List of Figures

List of Symbols

Chapter I

Introduction

It is widely accepted that wireless technologies and wireless sensor networks are very attractive for Industrial and SCADA applications (Gungor & Hancke, 2009; Vitturi, Tramarin, & Seno, 2013; Willig, Matheus, & Wolisz, 2005). In such systems, there is often a requirement to collect data periodically and reliably from a set of sensors distributed over a geographical area. Continuity in servicing real-time data in these systems is expected and the presence of channel errors can badly affect it. It is impossible to provide guarantees for perfect delivery reliability or worst-case packet transmission times on wireless channels but protocols and applications can be designed to handle some loss levels. The imprecise notion of "some loss levels" means, that while there are some loss patterns tolerable by applications, other patterns might not be accepted.

In this work, we adopt the approach of (m, k)-firm deadlines (Hamdaoui & Ramanathan, 1995, 1997) to clearly distinguish the allowed loss patterns from the forbidden ones. The integers m and k are defined as $0 \leq m \leq k$ and (m, k)-firm deadline is met, when at least m out of a window of k consecutive source frames have been successfully received at the destination. If we cannot meet the (m, k)-firm deadline within a predefined time window it is implied that *violation* has occurred.

One important obstacle for meeting deadlines is channel errors. This is especially relevant for wireless channels since these are known to be time-varying and possess high error rates (Willig, 2005). We restrict our examination to two channel models: a static one with a predefined packet error rate and a time-varying artificial stochastic channel, known as Gilbert-Elliot (Elliot, 1963; Gilbert, 1960). The latter is a special case of the family of

Hidden Markov Models (Rabiner, 1989).

We explain the system in the context of a base station that periodically polls a group of reachable sensors (known as *remotes*). Response packets are generated by all the remotes and transmitted over multiple uplink channels with independent properties, back to the center. Our system is time slotted; one packet fits into a slot and all packets have the same size. In addition, we define an uplink super frame with a constant structure that determines the poll period. Every super frame comprises a few slots for data packets and the remaining are reserved for retransmissions. Finally, the uplink's time might be interpreted as a sequence of super frames.

We focus on policies which target reliability in centralized uplink scheduling. Because data packets sent by remotes could arrive corrupted at the base, the base station may decide to request retransmissions in order to recover as many unsuccessful transmissions as possible. Therefore, a base station scheduling policy is required to make decisions that account for numerous losses from multiple remotes with respect to limited bandwidth. Additionally, we assume that the downlink transmissions (performed by the base) contain no errors.

We studied three scheduling policies and compared their performance by simulation. We evaluated how well these policies manage to mitigate losses within a known population of remotes. The goal was to reduce the long-term average violation of system's (m, k)-firm deadlines.

The scheduling in this thesis was focused on retransmissions. We designed a policy, that takes in account both the wireless channel properties and dis-

tance to violation[1]. Our hypothesis was, that such policy can be built based on *Reinforcement Learning* theory (Sutton & Barto, 1998) and outperform the other two baseline policies, which take only one of these two parameters into consideration.

One baseline algorithm, that we used, is called DBP. It evaluates stream's (m_i, k_i)-firm distance from violation. This information is then used to prioritize, for which stream the DBP would prefer to allocate retransmission slots. The other baseline algorithm (called CAOS) is exclusively concerned with the estimated channel qualities and preferentially allocating more slots to channels with higher packet error rates.

Using simulation, we performed multiple experiments over both the Static and Markovian[2] channels. In addition, we looked at scenarios with different uplink channel properties – investigating both homogeneous and heterogeneous[3] configurations. While we found that the learning scheduler is superior to the DBP, it was not able to overcome the CAOS policy and lost to it in the majority of examined scenarios.

The thesis is structured as follows: Chapter 2 provides related background on methods that use retransmissions in communications, (m, k)-firm deadlines, RL theory and surveys the related work. Chapter 3 describes the system under consideration more precisely. In Chapter 4, we expand upon the detailed design of our learning scheduler. The design of two baseline scheduling policies is presented in Chapter 5. Chapter 6 discusses the simulation details. Chapter 7 discusses the results. Finally, the conclusions and

[1] Defined in Equation 3.4 on page 43.

[2] Earlier referred to as Gilbert-Elliot channel.

[3] Multiple uplink channels of diverse qualities.

future work remarks are given in Chapter 8.

Chapter II

Background

2.1 Retransmissions in Real-Time Critical Data Applications

In this thesis, we develop and study policies that aim to enhance reliability with the method of retransmissions over wireless channels. In this chapter, we explain this method in detail and outline its common usages in wireless communications.

There are multiple methods to recover from errors, one of which is by retransmission. Retransmissions are a method of resending lost packets upon a request by the recipient. A receiver should first decide it is missing a packet or got a corrupt packet, and then informs the sender by issuing a retransmission request. The required bandwidth occupied by retransmissions usually grows if the amount of lost packets in a network goes up. Retransmission methods could be applied at different communication layers but our focus is on link-state retransmissions. In the discussion, we assume bi-directional communication.

Link-state retransmissions are often performed in the Medium Access Control Layer (MAC) and are also known by a more general name: Automatic Repeat Requests (ARQs) (Garcia & Widjaja, 2004; Lin, Costello, & Miller, 1984). ARQ methods are widely used to achieve reliable communication between computers. Retransmissions are known as time-bounded processes, which are protected by timers. Repeated packets become irrelevant with the expiry of an attached timer.

The sender's retransmissions are triggered by acknowledgments from the receiver. One type of acknowledgment is an Explicit Acknowledgment (eACK) (Mahmood, Seah, & Welch, 2015) that involves a transmission of special control message from the receiver assuring the sender that its packet was received

correctly. Another way of informing the sender is by a Negative Acknowledgment (NACK). The NACK is used when the received packet is corrupted and the address information is correct, i.e. receiver did not get the packet by mistake. The usage of eACK/NACK mechanisms can carry a large communication overhead. Furthermore, Mahmood et al. (2015) discuss the option of combining NACK and eACK to provide sender with a feedback from every packet or from a group of packets.

A non-MAC example of reliability implementation using ACKs is a Transport Control Protocol or TCP (Kurose & Ross, 2002). TCP employs retransmissions as part of its reliability strategy. Standard wireless technologies such as Bluetooth, IEEE 802.15.4 and IEEE 802.11 a/b/g also employ retransmissions (Willig et al., 2005). Moreover, LTE employs quality of service on bearers and depends on retransmissions which are performed at a radio link control layer (3GPP, 2008).

2.1.1 Retransmissions in Wireless Sensor Networks

Most of WSN data transport protocols that cover reliability are focused on uplink delivery guarantees (S. Kim, Fonseca, & Culler, 2004; Medidi, Nandanavanam, & Medidi, 2011), due to the more challenging nature of the uplink compared to the downlink. Sensors that deliver data to a sink, usually use a smaller transmit power (S. Kim et al., 2004) and their transmissions are potentially more sensitive to wireless channel interferences (Tse & Viswanath, 2005). In many cases, the uplink is a shared medium, so it is contention-based. This potentially reduces the chance of packets to successfully reach their destination (Tasaka, 1986). Another limitation of many WSN's is their

narrow bandwidth (S. Kim et al., 2004), little computational power and memory. Therefore, an execution of complex algorithms on sensors hardware in order to enhance their reliability may be of an issue.

Mahmood et al. (2015) classify retransmissions in WSN's into the categories of *end-to-end* and *hop-by-hop*. The end-to-end requires only the source packet node to retransmit the lost data, while in hop-by-hop the intermediate nodes are allowed to perform local caching of the lost information. In their review, Mahmood et al. (2015) scope numerous reliability protocols for data transfer in WSN and there is no doubt that the subject of reliability carries a major importance in Wireless Sensor Networks.

2.1.2 Retransmissions in Industrial Communication Networks

Industrial Communication Networks (ICN) are another area where the reliability of data transportation is considered as a central issue in the design of wireless solutions (Willig et al., 2005).

Some important technologies in industry automation include *fieldbus* systems, which use protocols like wired PROFIBUS (Tovar & Vasques, 1999), interconnecting digital controllers with other controllers and/or with field devices such as sensors and actuators. The emerging trend in ICN is the migration towards *wireless fieldbus* using technologies like wireless PROFIBUS (Willig, 2003), wireless HART (D. Chen, Nixon, Han, Mok, & Zhu, 2014) or ISA.100 (Radmand, Talevski, Petersen, & Carlsen, 2010). Fieldbus traffic is known to be recurring, periodic and/or acyclic (e.g. alarms) with bounded latency and jitter (Decotignie & Pleinevaux, 1993). It is subject to deadlines and thus has a nature of a real-time critical traffic. With the introduction of

the wireless fieldbus it is much harder to meet timing requirements and satisfy a similar reliability level as a wired network, given the adverse problems of a wireless channel.

2.1.3 Retransmissions in Multimedia Applications

Reliability in audio applications is often improved by receiver-side techniques like insertion, interpolation, regeneration and/or repetition, or by sender-side techniques such as retransmissions, interleaving or forward error correction (Chua & Pheanis, 2006; Maheswari & Punithavalli, 2009; Perkins, Hodson, & Hardman, 1998). For sender-based audio transmission recovery, Chua and Pheanis (2006) considered Redundant Data Transmission (RDT) in which one IP packet contains the previously transmitted and new audio data in the payload. The authors examined loss-recovery also by sending duplicate packets of the original copy. Chua and Pheanis (2006) found these methods more suitable for audio applications than the loss-dependent retransmissions, despite the introduction of constant addition to bandwidth and embedded delay.

In order to use retransmissions of lost audio/video packets, the receiver application must be able to support the latency caused by repeating packets. Multimedia traffic retransmit timeouts are often constrained in terms of latency and packet reception jitter (Gibson, 2001). Sze, Liew, and Lee (2001) considered retransmissions loss and gap detection methods to determine if the time to salvage the lost data has passed. Further improvements in reliability for multicast traffic were achieved by Nonnenmacher, Biersack, and Towsley (1998), who combined between retransmissions of lost data and

FEC. Such integrated schemes are also known as Hybrid ARQ in literature (Lin et al., 1984). Lu, Steenkiste, and Chen (2007) proposed an adaptive retry scheme for MPEG-like video streaming in wireless LANs in which they determine whether to discard or resend a packet, based on its retransmission deadline.

2.2 Scheduling Retransmissions

There are various scheduling policies that handle allocation of data or/and of retransmissions for multiple sources. The simplest to think of is Round Robin (or RR) (Ramabhadran & Pasquale, 2006; Saha, Mukherjee, & Tripathi, 1998). The allocation of packets to the variety of streams in RR, is done in circular order and in equal portions.

Chang and Hsiao (2015) propose to allocate retransmissions to various streams by a priority, defined by using *accumulated utility loss* and defined as $g_i u_i$ for traffic type i. A distance, g_i, is defined as the difference between the number of packets that should be received and the actual number of received packets. Larger values of the weight, u_i, attract more retransmission resources to traffic i. The complete definition for priority is $[u_i g_i]^{f_c}$, where f_c is a control factor responsible for adjusting fairness among streams, whilst $f_c = 0$ makes it work like RR. Moreover, the total amount of resources required for retransmissions may not be sufficient, so, it is normalized to meet the known available bandwidth.

Gamba, Tramarin, and Willig (2009) propose several retransmission policies for a centralized cyclic polling communication system over a wireless channel for ICN. One baseline technique, referred to as the *bounded imme-*

diate retransmission or BIR, makes up to a maximum of W transmission attempts to deliver a packet. The sender expects to receive an immediate response after each attempt, but if no acknowledgment is received the service is considered failed for the current cycle. UIR or *unbounded immediate retransmission* is very similar to BIR with the distinction that the maximal number of attempts in it is unlimited. QR or *queued retransmissions* is an enhancement of UIR that works as follows. All retransmit requests are held in a queue and the failed nodes are handled on a packet-by-packet basis. Pulling node i from the queue generates packet's transmission towards node i. If the recent trial fails, i's request is re-added into queue's tail. Note, that i is not added back if the opportunity window for that retransmission has expired. Similar to UIR, there are no limitations for the number of trials per node. The *Adaptive* QR, or AQR, is an enhancement to QR. In AQR, the long-term history of retransmissions successes and failures per node are recorded. Nodes with the largest number of successful trials are executed first. Immediate and enqueued retransmission strategies were also studied by Demarch and Becker (2007) for firm real-time traffic in 802.11e standard.

Scheduling policies, which focus on systems with real-time traffic, are also relevant. Willig (2005) evaluated the performance of several scheduling policies over Markovian channels subject to the (m, k)-firm deadline QoS constraints (see Section 2.3.3). These policies are based on a stream's distance to violation (defined in Equation 3.4 and denoted as d_i for stream i) and on a non-zero cost C_v that is incurred from a violation. Closest-to-violation-random (or CTV-R) policy serves streams that have the smallest distance to violation, whilst breaking ties randomly. A similar policy Closest-to-violation

Highest-Cost (or CTV-HC) selects streams with the largest C_v value first. Another policy, known as $L-(w_d, w_c)$, assigns the weighted average of d_i and C_i to stream i. Out of N streams, it selects the stream,

$$\arg\max_{i\in\{1,\ldots,N\}} \left(w_c \cdot C_i + \frac{w_d}{d'_i} \right) \tag{2.1}$$

where $d'_i = d_i$ if $d_i > 0$, otherwise $d'_i = d_i - 1$. Furthermore, the author concludes, that none of the studied policies is better under all different channel conditions. Finally, the approach of *meta-scheduling*, which offers the use of more than a single scheduling policy, is proposed. It is based on the accumulated history of channels conditions. For example, given a history of transmission successes and failures in the last h time slots, a meta-scheduler works through a finite set of candidate policies and selects one policy which would generate the least penalty (or cost) if applied.

2.3 Weakly-Hard Real Time Scheduling

2.3.1 Real-Time Data Critical Constraints

The Quality of Service (QoS) for real-time data is usually strongly dependent on the data's timing performance, or more accurately, on timeliness. The data in a wireless channel is subject to random and sometimes severe corruption. Wireless communication channels possess many well known problems such as thermal noise, fading, shadowing, multi-path and other interferences. The corruption caused by the channel introduces latency and affects applications such as audio (Perkins et al., 1998), video streaming (Shaikh & Ahmed, 2009), SCADA (Gungor & Hancke, 2009) and several industrial applications (Vitturi et al., 2013). Unfortunately losses are inevitable, but the vast majority of real-time critical data types are known to deal well with some loss, e.g. audio streaming, where a technique such as error concealment (Perkins et al., 1998) can be used. The sensitivity to losses is significantly dependent on data application. In SCADA applications, multiple and consecutive packet loss events can be tolerated for long seconds whilst a relatively short loss in constant bit rate video streaming (Shaikh & Ahmed, 2009) might be disastrous.

2.3.2 Allowed Patterns of Loss

We employ a conceptual framework in order to distinguish between the allowed and forbidden (or non-feasible) loss patterns. This enables us to investigate the characteristics of a system which tolerates missing some deadlines over a finite time window, given a precise distribution. In particular, we develop our learning policy in the context of Weakly-Hard Real Time (WHRT)

scheduling theory (Z. Wang, qiong Song, Poggi, & Sun, 2002). Quite often, the WHRT is described in literature with the application of task scheduling in operating systems. Its further discussion will be applied to wireless communication systems, *mutatis mutandis*.

WHRT could be identified as an intermediate class between two well known scheduling theory classes: Hard Real Time (HRT) and Soft Real Time (SRT). Applications that utilize HRT do completely forbid losses. An example could be of a process or task execution deadlines. The ones that use SRT are expected to meet deadlines within some probability of success (Abeni & Buttazzo, 1999; Song, Koubaa, & Simonot, 2002). When one specifies WHRT constraints for real-time critical applications, an interesting subclass named the (m, k)-firm deadlines (Hamdaoui & Ramanathan, 1995) can be identified.

2.3.3 Properties of (m,k)*-Firm Deadlines*

The variables m and k are integers such that $0 \leq m \leq k$, and the (m, k)-firm deadline is met when at least m out of a window of k consecutive source frames have been successfully received at the destination. The traditional deterministic requirement, which states that all deadlines must be met, can be expressed as a $(1, 1)$-firm deadline (this is HRT). If an (m, k)-firm deadline is not met, we say that a *violation* occurs.

The problem of scheduling multiple real time data streams was addressed by Hamdaoui and Ramanathan (1995). The authors offered a scheduling policy named Distance Based Priority (DBP). The DBP can deal with multiple streams having different (m, k)-firm deadlines for task executions at a central

server (see Figure 2.1). The policy prioritizes streams by their distance from stream's deadline violation (refer to Equation 3.4).

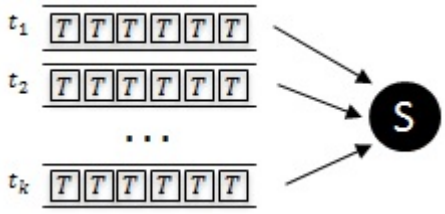


Figure 2.1: Multiple streams $t_1, \ldots, t_k$ served by central scheduler.

Following this work, a DBP-M scheme was proposed (Lindsay & Ramanathan, 1997) to support multi-hop networks extending the original DBP and providing end-to-end guarantees. While working solely based on distance to violation, the DBP scheme does not account for stream's timing characteristics (such as period or time in service) nor for its relationship to other streams. In order to resolve this, a DBP enhancement known as Matrix-DBP was proposed by Poggi, Song, Koubaa, and Wang (2003) for periodic streams. A scheme for aperiodic streams, named Equivalent Matrix, was proposed by J. Chen, Song, Wang, and Sun (2004).

Another novel DBP-based scheduling scheme, proposed by K.-I. Kim (2010), improves the distance of streams from violation by accounting for queuing delay while evaluating priority. K.-I. Kim (2011) and Li and Kim (2013) propose extensions for (m, k)-firm streams, which minimize packet deadline violations in multi-hop Wireless Sensor Networks. The metric of (m, k)-firm deadlines is used as another input to decide upon forwarding nodes. K.-I. Kim and Li (2012) also designed an (m, k)-firm based real-time congestion control scheme to improve the performance of their original algo-

rithm (Li & Kim, 2012) in terms of its control over source nodes transmission rates. The rate of stream i is evaluated using DTV (discussed in Section 2.3.3 further down) at the sink and reported back to its source for further sending rate adjustment.

Finally, in wireless fieldbus networks, (m, k)-firm deadlines were employed by Willig (2005). The author examined few scheduling policies over a range of simple channel models of varying burstiness. This method was used in the IEEE 802.15.4/Zigbee standard to provide a differentiation of services in beacon-enabled mode (Semprebom, Montez, Moraes, & Vasques, 2009).

Probability of Dynamic Failure

A stream that does not meet its deadline is considered to remain in violation (or to experience a dynamic failure). Hamdaoui and Ramanathan (1997) evaluated probabilities for dynamic failures using $(1, 3)$-firm deadline setting with 7 streams and an $M/M/1$ model. Packet inter-arrival times were exponentially distributed. An example of a Markov chain of a $(1, 3)$-firm deadline stream is shown in Figure 2.2. It describes all the possible states and transitions between states. Here, p_b denotes the transition probability of the following packet to be corrupted, while $1 - p_b$ represents a successful outcome. Note, that the probability for the following packet to fail (p_b) is itself a random variable and its value may change on every transition. Furthermore, the value of p_b may be drawn from an unknown distribution representing a real wireless channel. The state of some stream t is defined at specific moments as a tuple of k_t elements,

$$\left(\delta^t_{i-k_t+1}, \ldots, \delta^t_{i-1}, \delta^t_i\right) \tag{2.2}$$

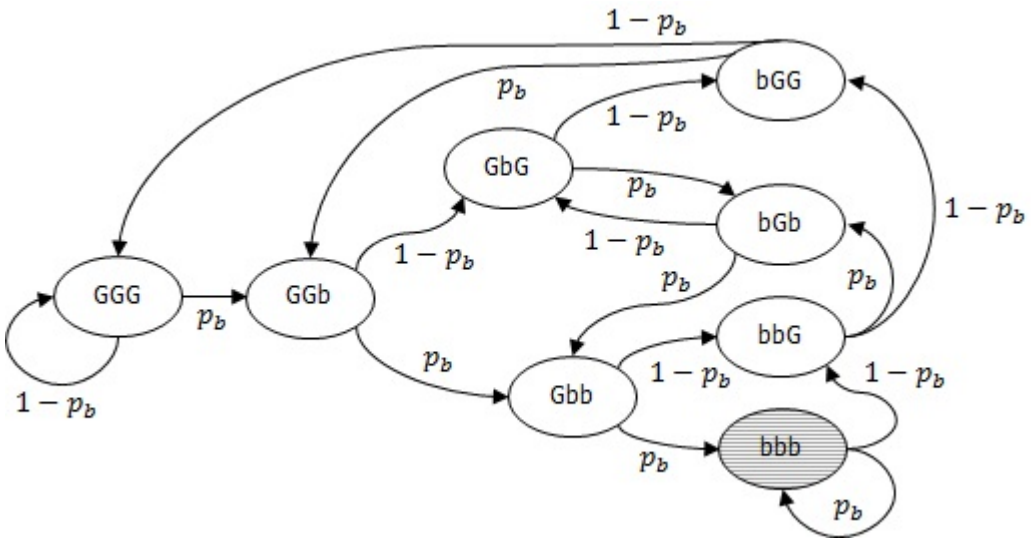


Figure 2.2: Markov chain of stream with $(1, 3)$-firm deadlines.

where i is the index of the most recent packet received in stream t. The stream can reside in one of 2^{k_t} possible states (eight states in case of $(1, 3)$-firm). States which have less than m_t packets within them are considered failure states. In Figure 2.2, good packets are denoted as G and the bad are as b. The state, in which the stream experiences a dynamic failure, is greyed out.

Distance to Violation and Stream Priority

The metric of distance to violation (DTV) was originally introduced by Hamdaoui and Ramanathan (1995). If the number of *good* packets (out of recent k_i consecutive packets) received in stream i is exactly m_i then for an (m_i, k_i)-firm deadline stream one can say the stream's DTV equals 1. The maximum distance to the failing state is $k - m + 1$. Hamdaoui and Ramanathan (1997) assign a priority $\rho \in \{0, 1, ..., k - m + 1\}$ to each one of the served streams

which corresponds to DTV. This is done in order to remove the streams that are less likely to violate deadlines. A stream in violation will be assigned $\rho = 0$. Stream are assigned with priorities according to their states (see Equation 2.2).

An example of priority evaluation for three parallel streams is demonstrated in Figure 2.3. The left side buffers, that can store up to four feedbacks, show the present receive state (G stands for Good or b for bad). The right side (from the arrow) shows the "next" feedback. Every new packet's feedback is pushed into the buffer from the right and all bits are shifted to the left.

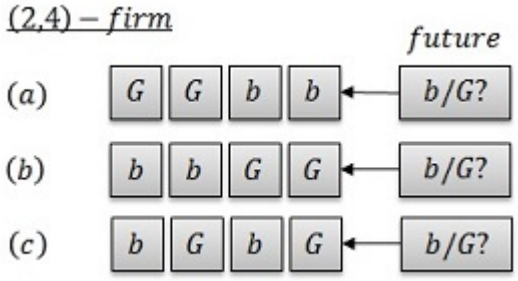


Figure 2.3: States that differ in DTV for $(2, 4)$-firm deadline scenario.

For simplicity, we assume that $Pr(b) = Pr(G) = 0.5$. Although the number of good and bad packets in each of the streams (a) to (c) is the same, their chances to experience a dynamic failure during the next state may be different. The next bad packet in (a) is expected to change buffer's state to {Gbbb}, i.e. to reach violation. In (b) however, its impossible to reach violation at the next step because both possible future states {bGGb} and {bGGG} are safe. The case of (b) is similar to (c), where deadlines can't be violated because the expected future states can be {GbGG} or, at the

worst case {GbGb}.

Hamdaoui and Ramanathan (1995) define the method of stream priority evaluation (the core of the DBP policy) as follows: the position of the y-th successful packet (G) from the right is defined as $l_t\,(y, s)$ for stream t and state s. If $y < m$ (number of good packets less than m) in s then $l_t(y, s) = k_t + 1$. The priority that is assigned to packet $i + 1$ in stream t is given by,

$$\rho_{i+1}^{t} = k_t - l_t(m_t, s) + 1 \tag{2.3}$$

recalling that state s_t is specified in Equation 2.2. Note that instead of the notation of ρ^t we sometimes use $\rho\,(t)$ or ρ_t. A substitution of 1 for G (Good) and of 0 for b (bad) for numeric evaluation of priorities is used in Chapter 5.2 in design of DBP-like baseline scheduling policy.

Limitations of DBP Scheduling

As noted by J. Chen et al. (2004), DBP ignores the stringent characteristic (easier/harder) of different (m, k)-firm deadlines. For example, in streams t and t' with $(9, 11)$ and $(3, 5)$-firm deadlines, with "01110111111" and "10011" states respectively, the DTV is 1. Nonetheless, it is harder to meet deadlines of the former stream t. A DBP scheduler does not take this knowledge into account and would evaluate $\rho\,(t) = \rho\,(t')$. This required distinction was established in the work of Ramanathan (1999).

When a stream is in a failed state, there are still numerous state combinations possible. For example, the violation states of a $(3, 5)$-firm stream are {GbbbG}, {bbbGG}, {bbbGb}, etc. The issue with that is, that DBP would generate the same priority, ρ, for all in-violation states even though

there could have been a value, had the above been treated unequally.

Finally, streams can have properties, for example, message size, channel quality, special timing requirements or periods, which differ between streams. DBP takes a local view when making the decision upon priority and therefore it may be suboptimal from a system's perspective. Improvements along some of these lines were made by Poggi et al. (2003).

2.4 Reinforcement Learning

Reinforcement Learning (RL) (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998) deals with systems that at each time can be in one of a finite (but possibly large) number of states, and each state s is a member of the state space $\mathcal{S}$. This theory is focused on learning towards a predefined goal. The experience gained from learning is by pure interaction with *environment* through reception of numeric rewards[1]. Suppose that a RL agent starts at some initial state, s_0, and performs its first action, denoted as a_0 (an action a from a feasible action space denoted as $\mathcal{A}$), which moves the agent's state to s_1 and derives the reward r_1. From state s_1, the agent may choose an action a_1, moving it into state s_2 with reward r_2, and so forth (see Figure 2.4).

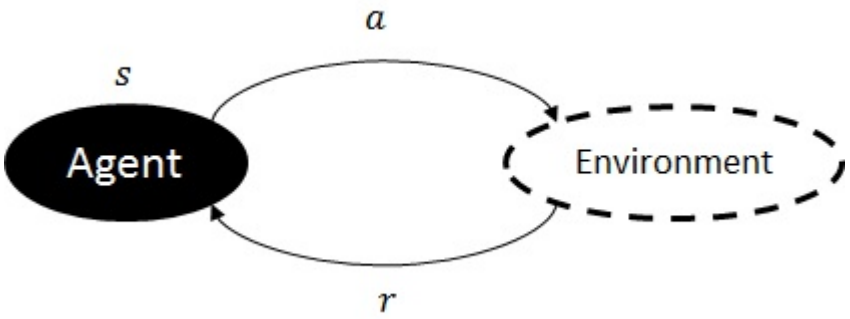


Figure 2.4: The general scheme of RL agent's learning process.

The experience gained from the process, described above, is translated to values inside agent's memory. This information contributes to the decisions on future actions, $a_3, a_4, ...$, while following the rationale of maximizing the rewards, that the RL agent receives over time. In order to accomplish this, the agent must be sensitive (to some extent) to the state of an environment

[1] Reward is denoted as r and quite often $r \in \mathbb{R}$.

it interacts with. Moreover, it must be able to select actions which influence the state towards higher rewards.

The environment in Reinforcement Learning is typically formulated as a Markov Decision Process (MDP). Using this formulation, an agent's states do not have to remember the whole history of their past sensations. If we assume that an agent has Markov property, it allows the history to be stored in a very compact form using,

$$\begin{aligned} Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_0, a_0, r_1, \ldots, a_{t-1}, s_{t-1}, r_t\right\} \\ = Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\} \end{aligned} \tag{2.4}$$

where all history i.e. the sequence $s_0, a_0, r_1, \ldots, a_{t-1}, s_{t-1}, r_t$ is accounted for within s_t and a_t. Note that the reward r_{t+1} is the result of taking action a_t in state s_t. The state transitions are governed by a time-homogeneous Markov chain (Norris, 1997) – when in state s some action a is chosen, the transition probability from s to s' can be written as,

$$p(s'|s, a) = Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \tag{2.5}$$

Furthermore, an immediate reward is defined as,

$$r(s, a) = E\left[r_{t+1} \mid s_{t+1} = s', s_t = s, a_t = a\right] \tag{2.6}$$

In RL, agents may follow policies which assign to each state $s \in \mathcal{S}$ a suitable action $a \in \mathcal{A}$. The latter can happen either deterministically or randomly. The policy is then given by a probability distribution over the set of actions. A policy is denoted as π and an action performed using policy π from state

s is denoted as $\pi(s)$. Following some policy π_1, an agent can yield a specific return over its future steps. Conversely, shifting to an another policy π_2 might affect the value of expected future rewards. For some initial policy π_k and a given starting state $s_0 \in \mathcal{S}$, the system evolves through a sequence of states $s_1, s_2, s_3, \ldots$ and chooses actions $\pi_k(s_0), \pi_k(s_1), \pi_k(s_2)$ an so on. The expected returns for state s, when following a policy π, can be formulated for an MDP as a state-value function,

$$V^{\pi}(s) = E_{\pi}\left(R_t \mid s_t = s\right) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right) \tag{2.7}$$

and hence V^{π} is also the notion of "how good" it is to be in state s. The R_t is the reward expected to be received in the future and $\gamma \in [0, 1)$ is a discount factor reducing the value of future rewards. As an alternative to calculating $V^{\pi}(s)$, one can employ the notion of "how good" it is to execute an action a in state s by using an action-state value function denoted as Q^{π}. In the case, where the $p(s'|s, a)$ and $r(s, a)$ (see definition in Equations 2.5 and 2.6) can be found, the function $V^{\pi}(s)$ is sufficient because a look of one-step ahead would provide the best combination of the next state and reward. Quite often, however, the environment's properties are unknown *a-priori* and only the evaluation of all actions would be sufficient to suggest a policy. The state-action value function, when following policy π, is defined as,

$$\begin{aligned} Q^{\pi}(s, a) &= E_{\pi}\left(R_t \mid s_t = s, a_t = a\right) = \\ & E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right) \end{aligned} \tag{2.8}$$

Here the R_t is associated with first taking action a in state s and following

policy π thereafter. Equations 2.7 and 2.8 are a discounted formulation of the value and action-value functions. In the case of RL with episodic tasks, where the agent's learning process breaks into finite episodes, one would use an undiscounted formulation with $\gamma = 1$.

As follows from MDP properties, an agent's search during the process of learning can be restricted to *stationary* policies for which the distribution of their generated actions depends only on the last visited state. It may even be restricted to stationary policies where the action selection is deterministic based on the current state.

The desired outcome of the learning process is an increase in returns over time. In order to achieve that, the policy followed, ought to be replaced with other policies that yield a better result. An optimal policy is denoted as π^* and provides the highest return. One or more optimal policies may exist amongst all stationary policies if $\mathcal{S}$ is finite and countable. Rewards are bounded and transition probabilities don't vary between decision epochs (Puterman, 1994). The optimal state value function and the state-action value functions are denoted as V^* and Q^* respectively.

The Reinforcement Learning methods that we will present do not assume a complete knowledge of the environment. Their behavior is purely based upon raw experience. Before proceeding, we will introduce the two important concepts of Generalized Policy Iteration and Exploration.

2.4.1 Generalized Policy Iteration (GPI)

Almost all Reinforcement Learning methods can be described as Generalized Policy Iteration (GPI). The general idea of GPI (Sutton & Barto, 1998)

is to have an interaction between the policy improvement and evaluation processes. Policy improvement process makes the policy greedy[2] with respect to the current Q^{π} function, while the evaluation process makes the value (or action-value) function consistent with the current policy. Such interaction[3] is expected to result in convergence to an optimal function $Q^{*}(s,a)$ and to the policy that provides the maximal return – the π^{*} policy.

Suppose that some RL algorithm begins with an arbitrary policy π_0 and some value function $Q^{\pi_0}(s,a)$. The goal is achieved by a gradual process,

$$\pi_0 \rightarrow Q^{\pi_0} \Rightarrow \pi_1 \rightarrow Q^{\pi_1} \Rightarrow \cdots \Rightarrow \pi^{*} \rightarrow Q^{*} \tag{2.9}$$

where $\rightarrow$ represents policy evaluation and $\Rightarrow$ represents policy improvement. In finite MDPs, the number of policies is $|\mathcal{A}|^{S}$. If the Generalized Policy Iteration is used a convergence to the optimal policy after a finite number of steps is guaranteed. The coupling between evaluation and improvement does not have to be strong in order to achieve convergence. In some cases, only partial states are updated before the shift from evaluation to improvement but the algorithm's performance is not affected.

2.4.2 Balancing Exploration

In order for an RL agent to converge to an optimal policy, it should have the ability to identify optimal actions within a given action space $\mathcal{A}$. Balancing exploration and exploitation is important; an agent may find a *good* policy

[2] By that, the policy itself changes.

[3] GPI includes both the elements of competition and cooperation. Its processes compete by pulling in opposing directions but manage, by cooperation, to accomplish a single solution.

to follow but a better one may exist. It is still possible, however, that it might stay unreachable without a *sufficient* exploration. On the other hand, too much exploration might undermine agent's learning, which is based on the exploitation of recently acquired knowledge. Finding a proper level of exploration is one of the fundamental problems in Reinforcement Learning (Hester, Lopes, & Stone, 2013; Lopes et al., 2012; Singh, Jaakkola, Littman, & Szepesvári, 1998; Sutton & Barto, 1998; Tokic, 2010).

Suppose that an agent follows a greedy policy. In other words, it exploits its current knowledge by the continuous selection of some action a' in state s, for which $Q(s, a') = \max_a Q(s, a)$[4]. In that case, the agent has no way of trying other inferior actions, which can potentially lead to a better outcome. One method that allows the agent to stay greedy most of the time and still be able to explore, is called *ϵ-greedy* (Sutton & Barto, 1998). Following this method, an agent selects greedy actions with probability $1 - \epsilon$, otherwise, a random action is chosen with a *smaller* probability ϵ.

In the ϵ-greedy method, both the worst and the next-to-best actions can be chosen with a similar likelihood. Alternatively, we may define action selection probabilities as a graded function of the estimated value, i.e. the value of $Q(s, a)$. This idea is called the *softmax* (Busoniu, Babushka, & Schutter, 2010; Schaerf, Shoham, & Tennenholtz, 1995; Sutton & Barto, 1998) method and its based on Gibbs (Boltzmann) distribution,

$$P(a|s) = \frac{\exp(\tau Q(s, a))}{\sum_b \exp(\tau Q(s, b))} \tag{2.10}$$

[4] The estimated action-value is denoted as $Q(s, a)$; no information on how it was acquired (following a policy or not).

where $P(a|s)$ is a probability to choose action a while in state s. $\tau > 0$ is a given system parameter, often referred to as the "system temperature". For small values of τ, Equation 2.10 allows a system to pick non-optimal actions more frequently. Conversely, an optimal action may be regularly selected for large values of τ. The relationship between exploitation and exploration can therefore be controlled by tuning τ.

We will expand the discussion over exploration control in Section 2.4.5, however first, we should discuss some central methods of Reinforcement Learning theory.

2.4.3 Monte Carlo

There is a method called Monte Carlo (MC), which learns by averaging returns of acquired values. The learning of an MC agent is divided into finite episodes; all averages are updated at the end of each episode. This method removes the need to know $p(s'|s,a)$ and $r(s,a)$[5] because it is based only on experience – sample sequences of rewards (Sutton & Barto, 1998). Eventually, states and actions are updated as a result of agent's interaction with an environment. Additionally, estimates and policies can be replaced at each episode. One averaging method that is employed by MC agents and called the *every-visit MC*. This returns the average for every time some state s is visited in an episode. Another method, known as *first-visit MC*, generates the average the first time that s is visited in an episode. The Monte Carlo method has an asymptotic convergence to an optimal value function for both the first and every-visit methods. The downside of this algorithm

[5] Attributes of environment, defined in Equations 2.5 and 2.6.

is it's convergence time, which is expected to grow if the chosen episodes are lengthy.

Monte Carlo employs the principles of GPI[6] and determines Q^* using only its sample experience. Within the followed policy, the agent must allow some level of exploration during the process of actions selection. This stimulates reaching valid comparison between all values in a state when making predictions about the future. Another strategy is to run *off policy*. In that case, one still uses a policy in order to generate an algorithm's *behavior*. However, this "behavioral policy" is unbiased from the estimation of values. Furthermore, an *estimation* policy may also exist and it is allowed to be greedy, while the previous one deals with exploration (Sutton & Barto, 1998). Both the off-policy and *on-policy* strategies can be applied to MC, but also to TD-learning algorithms, which we cover next.

2.4.4 Temporal Difference Learning

There is another RL method called Temporal Difference (TD), which is not required to know the dynamics of the environment. Similar to MC, a TD agent also learns from raw experience. TD-learning, however, is not episode-based like MC. This method constructs estimates based on previously learned information during every step it makes. This enables a TD agent to change predictions very quickly; unlike an MC agent that would have to wait until the end of the current episode in order to make decisions.

In TD-learning, an agent chooses an action a_t at the current time t and is capable of observing the reward and new state at the next time $t + 1$.

[6] Explained in Section 2.4.1.

Moreover, this agent updates the current state's value function $V(s_t)$[7] immediately after the reward's reception. TD(0) is known as the simplest version within the TD-learning algorithms family. In it, value updates are performed using,

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V\left(s_{t+1}\right) - V(s_t) \right] \tag{2.11}$$

where r_{t+1} is a reward from the selection of a_t from s_t. The discounted element $\gamma V\left(s_{t+1}\right)$ is the predicted impact on state-value $V(s_t)$, because of the move to state s_{t+1}. Finally, $\alpha \in (0, 1]$ is a small and constant *learning factor*. In the family of TD-learning methods, the TD(0) learning is a specific case that accounts for only one future step. Conversely, a generic TD(λ_e) assumes a component of history back-trace, in which $\lambda_e \in [0, 1]$ is a trace-decay parameter. All previous returns are weighted by,

$$R_t^{\lambda_e} = (1 - \lambda_e) \sum_{k=1}^{\infty} \lambda_e^{k-1} R_t^{(k)} \tag{2.12}$$

where factor $(1 - \lambda_e)$ weights the last step return, $(1 - \lambda_e)\lambda_e$ weights the step before the last, and so forth. The value of λ_e indicates to states to which degree they can vary as a result of learning. In summary, the parameter λ_e adjusts the level of influence of outdated samples.

Generally speaking, TD methods are advantageous for tasks in which the environment is quickly changing compared to the episodic MC. Next, we will discuss a couple of TD(0) algorithms: one called SARSA and the other Q-Learning.

[7] In TD-Learning we simplify the notation of $V^{\pi}(s_t)$ to $V(s_t)$ because we discuss both the on and off-policy learning methods.

2.4.5 TD On-Policy and Off-Policy Learning

One central algorithm for learning an MDP policy is called SARSA. It adjusts the $Q(s, a)$ functions using the action-value *update rule*,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{2.13}$$

where $Q(s_{t+1}, a_{t+1})$ is dictated by a policy, since action a_{t+1} is selected by an underlying policy at the next state s_{t+1}. This one-step learning equation is symbolized by the quintuple $\langle s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} \rangle$[8]. It generates the transition from $\langle s_t, a_t \rangle$ to $\langle s_{t+1}, a_{t+1} \rangle$ with reward r_{t+1} as the result.

An alternative approach to SARSA can be an off-policy learning method. The most popular method in this respect is Q-Learning (Watkins, 1989). It uses the one-step update rule,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in A_s} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.14}$$

Q-Learning estimates the optimal state action value function Q^* directly. Disregarding the selected and executed action with respect to some policy π[9], the experience updates occur using the *most valued action* from a finite set of values $Q(s_{t+1}, a)$. It follows that a Q-Learning agent chooses optimal actions as estimated as opposed to SARSA, which looks at the explored action before making the next move. SARSA learns to be cautious in environments where the exploration is costly, unlike Q-Learning. In the context of the *Cliff World*

[8] The word SARSA originates from this tuple.

[9] This part belongs to a behavioral policy, mentioned in Section 2.4.3.

example by Sutton and Barto (1998), in the case that an agent chooses the wrong action it will fall off the cliff, unless it found a longer path away from the abyss. In this example, taking wrong actions in the latter case won't hurt as much. Q-Learning takes the cliff path, since it is indifferent to the costs incurred by exploratory actions. Conversely, SARSA learns to take the slow path because it gains experience on a cost that exploratory action may incur. Nonetheless, if we gradually reduce exploration, both SARSA and Q-Learning would asymptotically converge to the optimal policy.

2.4.6 Pseudo Code of SARSA and Q-Learning

Algorithm 1 outlines the code of SARSA and Algorithm 2 outlines the Q-Learning code. Both are TD(0)-learning algorithms and are the original algorithms from the book by Sutton and Barto (1998), which are designed for the case of episodic tasks. In this thesis, we have simplified them to support a single episode of an infinite length. This is a better match to the learning policy which we have developed. Note that the SARSA update rule in Algorithm 1 conforms to Equation 2.13, while the update rule of Q-Learning in Algorithm 2 conforms to Equation 2.14.

Algorithm 1 $TD(0)$ $SARSA$

1. $Q(s, a) \leftarrow arbitrary$
2. initialize s
3. **for** (each step from $1, \ldots, \infty$) **do**
4. Take action a_t; observe r_{t+1} and s_{t+1}
5. Pick a_{t+1} from s_{t+1} using policy derived from Q (e.g. $softmax$)
6. $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$
7. $a_t \leftarrow a_{t+1}$; $s_t \leftarrow s_{t+1}$
8. **end for**

Algorithm 2 $TD(0)$ Q-Learning

1. $Q(s, a) \leftarrow arbitrary$
2. initialize s
3. **for** (each step from 1 $, \ldots, \infty$) **do**
4. Choose a_t from s_t using policy derived from Q (e.g. *softmax*)
5. Take action a_t; observe r_{t+1} and s_{t+1}
6. $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, \mathbf{a}) - Q(s_t, a_t) \right]$
7. $a_t \leftarrow a_{t+1}$; $s_t \leftarrow s_{t+1}$
8. **end for**

2.4.7 Single Agent Convergence in TD

In the case where the learning coefficient α is sufficiently small, Equations 2.13 and 2.14 converge to Q^* (Sutton & Barto, 1998) with probability 1. This occurs only if the conditions by Robbins and Monro (1951) for learning factor α are met. These conditions are,

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{2.15}$$

In addition to this, the convergence for TD(0) SARSA (Singh et al., 1998) and for Q-Learning (Watkins & Dayan, 1992) is guaranteed if a level of exploration is such that all action-values are visited infinitely often. One central idea to RL convergence analysis is called "Greedy in the Limit with Infinite Exploration", or simply – GLIE. Following this, an agent gradually reduces the amount of exploration that it uses over time. Eventually it should retain a *minimal* level of exploration in order to keep the selected actions greedy in-the-limit with respect to $Q(s, a)$ functions. A utilization of this idea in a policy[10] secures the agent's convergence to the optimal values.

[10] See details in Chapter 4, Section 4.4.3.

Singh et al. (1998) provide bounds on the value of exploration parameters for on-policy learning; both for the ϵ-greedy and softmax methods. For example, a GLIE property can be utilized in softmax action selection by varying of the parameter τ. In this case, the function $\tau(t)$ decays with time (Boutilier, 1996)[11].

Additionally, authors like Hester et al. (2013) and Lopes et al. (2012) propose model-based exploration strategies, while Tokic (2010) offers exploitation, based on value differences, as a method to balance exploration.

In our learning policy, we consider elements that are inherent to the Multi-Agent Reinforcement Learning (MARL)[12]. The convergence in a case of a multi-agent scenario is no more straightforward, however, even in MARL systems GLIE policies can still deliver convergence (Claus & Boutilier, 1998).

2.5 Multi-Agent Reinforcement Learning (MARL)

MARL is an important field in Reinforcement Learning. It is applied in domains such as telecommunications and robotics (Busoniu et al., 2010). In the presence of many agents (where each is learning an MDP policy), the problem can generally be seen as a stochastic game (Shapley, 1953), or Markov game (Littman, 1994).

The multi-agent scenario is defined as a tuple, $\langle \mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{T}, r_1, \ldots, r_n \rangle$, where n is the number of agents, $\mathcal{S}$ is a finite environment state-space, $\mathcal{A}_k$ is a finite set of actions available to agent k, $\mathcal{T}$ represents all states transition probabilities and r_k is a reward function of agent k. The *joint action*

[11] Boutilier (1996) experienced with $\tau(t) = 0.995^t$.

[12] The motivation is explained in Chapter 4.

space is formed as $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, where the transition probabilities are $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and the reward function for agent k is $r_k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

It turns out that the Q function of one agent may be dependent on the decisions that other agents take and that the returns are correlated. During the game, agents always seek to maximize the total payoff i.e. the sum of all the individual rewards. In other cases, a reward can be shared, hence individual rewards aren't defined. Also note that the reward functions can be written as n-dimensional matrices. This is why these games are called Matrix games (Bowling & Veloso, 2000; Littman, 1994, 2001b).

We can differentiate between Markov *team* games (X. Wang & Sandholm, 2002) and *zero-sum* games (Littman, 2001a) for competing agents. Zero-sum games possess a unique Nash Equilibrium (NE). This is a point, in which any deviation from the present strategy would not improve the achieved payoff. Team games can converge to one out of many Nash equilibria.

2.5.1 Coordination between Agents in MARL

Stochastic games can be fully cooperative (or collaborative) (have common goals and identical reward functions) or competitive (have opposed reward functions), or mixed (Busoniu et al., 2010). Agents can be selfish[13] or conversely, have an embedded *coordination*. It is possible that they have neither, but have some *awareness* of others in the vicinity, which may be achieved via communication (Schaerf et al., 1995). In this context, it is important to also distinguish systems where multiple agents co-exist in one place. For example, a base-station that runs one agent per stream in order to perform packet

[13] Refers to definition by Bab and Brafman (2008) of 'non-cooperative' – lacks of any collaboration between agents except through the game.

scheduling. This is different from a decentralized concept of operation[14]. In practice, it is simpler to coordinate actions in a centralized system; however sensible social laws may improve efficiency in both constellations (Shoham & Tennenholtz, 1992).

Kapetanakis and Kudenko (2002) offer the FMQ algorithm, which improves the coordination in cooperative MARL by tracking information on how often actions generate maximal rewards. Huang, Yang, and Liu (2005) deal with the reduction of complexity for coordination tasks by a team of agents.

2.5.2 Best Response and MARL Convergence

Due to the simultaneous learning of numerous agents, a single agent may identify the environment as non-stationary, i.e. non-Markovian. This means, its convergence assumption no longer applies like in the single RL agent's case. A performance of one Q-Learning agent in terms of convergence may be recognized only as the "best-response" to other agents actions, with the assumption that the rest have all converged to stationary policies (Bowling & Veloso, 2000, 2002).

An algorithm called Minimax-Q (Littman, 2001b) is a version of Q-Learning for MARL. In this method, a team of agents would converge to NE if all state action-values are visited infinitely often. In Minimax-Q, a set of policies $\pi_1, \pi_2, \ldots, \pi_n$ is in state NE, if each one of them is the best response to the others. Bowling (2005) offers to measure the convergence regret (miss level) compared to the best static policy. Bowling and Veloso

[14] The majority of MARL research does not assume centralization.

(2001) offer an extension to Q-Learning proven rational[15] and convergent in MARL. Finally, Boutilier (1996) demonstrates convergence both for joint action learners (coordinated agents) and for independent learners (uncoordinated, operate Q-Learning in the classic fashion and ignore others).

[15] Rationality – if other agents policies converge to stationary policies then an agent is expected to produce the best response to others (Bowling & Veloso, 2001).

Chapter III

System Model

3.1 Network and Load Model

Firstly, we consider a star network consisting of one central base station and N remote stations or sensors (see Figure 3.1). This network operates using a TDMA-style protocol. More precisely, time is partitioned into consecutive superframes and each superframe is sub-divided into time slots. A superframe consists of three logical components: downlink, uplink and retransmission. In the *downlink* part, the base station broadcasts packets to all remote stations. In the *uplink* part, there is a separate time slot for each of the N sensors which they use for transmitting the freshly arrived packets for the first time. The *retransmission* part consists of another K time slots which are allocated for retransmission of uplink packets.

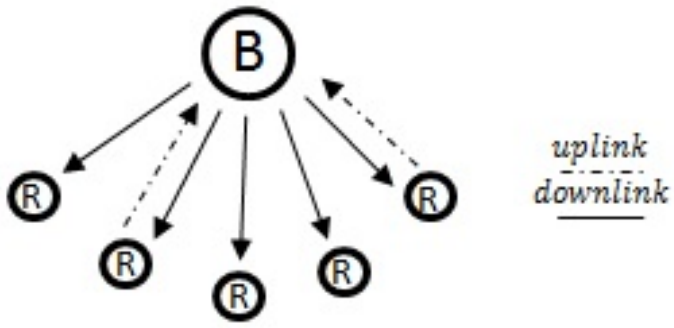


Figure 3.1: Star connection of one base-station to multiple remotes.

We do not impose any fixed order on these three parts. However we do assume that in the downlink part, the base station has the opportunity to announce how many of the K retransmission slots are allocated to each remote station. Moreover, the above announcement includes the exact order in which all transmissions and retransmissions should be executed.

In Figure 3.2 below, we depict a setup where we show the downlink (top line) and uplink's transmissions (bottom line) within a short time fragment. This drawing suggests the occurrence of a periodic downlink transmission, denoted as A, which is followed by a series of uplink packets[1]. In particular, the uplink's superframe users start retransmitting packets that were lost during the previous superframe. The retransmissions are denoted as R_i, where i is a stream's index. They are followed by multiple user data packets, denoted as T_i. Note that although the presented setup allows multiple data packets from the same user[2], in this thesis, we generate only one packet per user in a superframe, in order to simplify the analysis.

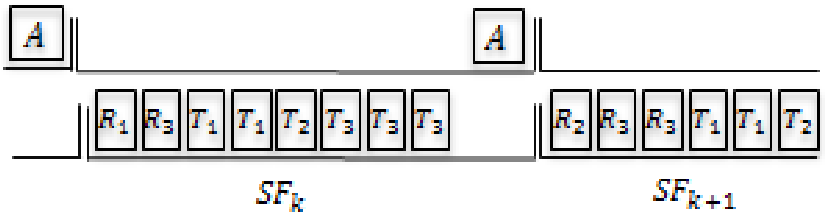


Figure 3.2: Transmission in superframes.

Also note that the above assumptions can be mapped to different technologies. This transmissions scheme is one of the options available in the *low latency deterministic network* (LLDN) extension to IEEE 802.15.4 (Anwar & Xia, 2014; F. Chen, German, & Dressler, 2010; IEEE Standard for LR-WPANs, 2012), which explicitly targets applications in the domain of factory automation. Another option offered by this standard (and commensurated with our model) is to start a superframe with a beacon on the first half of

[1] Which uplink packets will appear in SF_k depends on the downlink announcement done in SF_{k-1}, i.e. information out of the first A-packet in Figure 3.2.

[2] Note that packet T_3 appears three times in SF_k.

the downlink part, followed by the uplink part, followed by a group acknowledgment on the second half of downlink part and the retransmission part. WirelessHART (D. Chen et al., 2014) networks can also be configured to be compliant with our model.

Each sensor (or remote station) generates one new data packet at the start of each superframe. We assume that all these data packets have the same size and are protected by a strong checksum, i.e. any errors introduced by the channel are "perfectly" detected. Additionally, the packet headers have their own checksum so that we can avoid confusion due to an invalid addressing. We also assume that to each remote station i, a separate (m_i, k_i)-firm deadline is associated and all these deadlines are known to the base station. An individual packet generated by a sensor is said to have missed its deadline when it has not been received successfully by the base station at the time when the last possible retransmission slot for this packet has already passed.

3.1.1 Behaviour of a Base Station

The base station keeps track of two important pieces of information about each remote station i: the current packet error rate to station i, and the current distance to violation of the (m_i, k_i)-firm deadline of station i. The base station maintains an estimate of the current Packet Error Rate (PER) for the channel between station i and itself. More precisely, for each time slot assigned to station i in the uplink part or the retransmission part the base station observes whether it has received a packet correctly or not. This outcome for station i is denoted as o_i (with a packet error as $o_i = 1$ and a

successful reception as $o_i = 0$). The PER estimate $\hat{p}_i$ is passed through a first-order low-pass feedback filter (Hunter, 1986),

$$\hat{p}_i := (1 - \mu) \cdot \hat{p}_{i-1} + \mu \cdot o_i \tag{3.1}$$

Based on the results of a preliminary performance study, we fixed the parameter μ as $\mu = 0.1$. Note, that this type of estimator can naturally adapt to changing channel conditions, as it "forgets" older observations after some time.

In order to assess, how far a remote station i is away from violating its (m_i, k_i)-firm deadline, the base station maintains a sliding window of the last k_i transmission outcomes, where each outcome is represented by one bit. A transmission outcome in this context indicates whether or not the base station has somehow received station i's new data slot at the end of a superframe – either from a direct transmission or a retransmission. A successful outcome is assigned a bit value of one; a failed outcome is assigned a value a zero. If we denote the current sliding window of station i as $\mathbf{v}_i = (v_{i,k_i}, v_{i,k_i-1}, \ldots, v_{i,1})$ and the new outcome at the end of the retransmission slots as $v_{i,0}$, then the base station updates the state to,

$$\mathbf{v}_i' = (v_{i,k_i-1}, \ldots, v_{i,1}, v_{i,0}) \tag{3.2}$$

i.e. it "shifts in" the new outcome from the right. The "one-norm" of state $\mathbf{v}_i = (v_{i,k_i}, v_{i,k_i-1}, \ldots, v_{i,1})$ is given by

$$\|\mathbf{v}_i\| = \sum_{t=1}^{k_i} v_{i,t} \tag{3.3}$$

and it indicates how many of the recent k_i packets have not been received. Note that for convenience we have suppressed the time dependence of the state and the outcomes in our notation.

When $\|\mathbf{v}_i\| > k_i - m_i$ holds, we say that stream i is in deadline violation. We define the distance to violation d_i (or DTV) for stream i as

$$d_i = (k_i - m_i + 1) - \|\mathbf{v}_i\| \tag{3.4}$$

and note that for $d_i \leq 0$ stream i is in violation state.

At the end of each superframe, the base station updates the state $\mathbf{v}_i$ for all remote stations and also calculates their distances to violation. Furthermore, depending on the considered scheme, the base station uses this information and the estimated packet error rates to calculate an allocation of the K retransmission slots to stations. This allocation is then announced (by a broadcast) in the nearest downlink packet.

3.1.2 Behavior of a Remote Station

While the remotes are only responsible for traffic generation, the requirements for them are very simple compared to the base station. We assume that every remote receives the periodic advertisements from the base station. A downlink packet received during some superframe SF_k is used to evaluate remote's own transmission time offsets for the next superframe SF_{k+1}.

In a superframe, we allow K slots for retransmissions. Following K retransmission slots[3] the first remote will transmit user data packet on the

[3] Although the number of retransmissions might be smaller than K we still reserve the unused slots.

$(K+1)$-th slot, the second will transmit on the $(K+2)$-th slot, until the $(K+N)$-th slot, where $K+N$ equals the size of one superframe in slots.

Figure 3.3 outlines a more specific example of our system's superframe transmissions. This system is comprised of five remotes, i.e. $N=5$, $K=3$ and two consecutive superframes are presented.

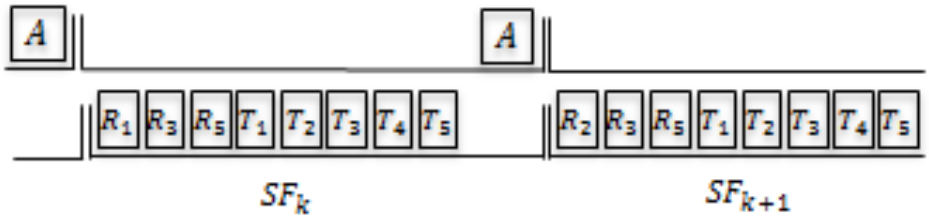


Figure 3.3: A pair of consecutive superframes in a 5-user system.

In superframe SF_k, the retransmissions were allocated to remotes 1,3 and 5, since the base station has experienced losses in supeframe SF_{k-1} from these nodes. Furthermore, the packets R_2,R_3 and R_5 in SF_{k+1} suggest, that the user data packets from remotes 2,3 and 5 have *not* been successfully received at the base station during superframe SF_k.

3.2 Channel Models

For our performance studies, we consider different channel models, each subdivided into static or dynamic models. In static models, the packet error rate of a particular channel does not vary over time. However, in dynamic channels it does. In all cases, there exists a separate and stochastically independent channel between each pair of nodes. The channels between different pairs of nodes can have different packet error rates (on average). We assume that only the data packets from remotes to the base station

are affected by errors whereas packets sent by the base station are received perfectly by all stations. This assumption is consistent with scenarios, where sensor nodes have to limit their transmit power, e.g. due to energy constraints (Stankovic, Abdelzaher, Lu, Sha, & Hou, 2003). That, in turn, may lead to non-negligible data loss rates (depending on the distance and other factors), whereas the base station can use a much higher power.

The first model is referred to as the static-homogeneous model. In this model, all channels lose packets independently of each other, with a fixed and common packet error probability p_{hom}. In the static-heterogeneous channel model, for each channel C_i between source i and the base station, a separate packet error rate p_i is assigned. Its values are manually generated and assigned by us, over the interval $[p_{het} - a, p_{het} + a] \subset (0, 1)$, where we denote by p_{het} as the chosen average packet error rate. The individual channels error rates that contribute to a value of p_{het} are all specified under the experiment's parameter details.

We have also chosen to work with a dynamic channel model to investigate the performance[4] of our learning-based scheme in the case of changing channel conditions. For an individual channel we use the Gilbert-Elliot (GE) channel model (Elliot, 1963; Gilbert, 1960) (see Figure 3.4). This channel operates in slotted time, with each time slot corresponding to the time slot of our underlying TDMA system. In each time slot, the channel is in one of the two states, figuratively called "good" and "bad". The state evolves according to a two-state time-homogeneous Markov chain with transition matrix $\mathbf{P}$. For each state, a separate packet error probability is assigned,

[4] in terms of minimizing violation; defined later in Section 3.3

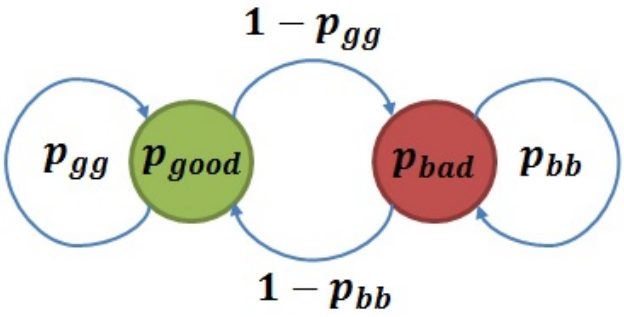


Figure 3.4: Markov discrete chain of the Gilbert-Elliot channel model.

which we denote as p_{good} and p_{bad}, respectively. When the channel is in state $s \in \{\text{good}, \text{bad}\}$, at the beginning of the k-th packet transmission, then the packet is transmitted correctly with $1 - p_s$ and erroneously with p_s. The packet transmissions in different slots are independent. At the end of this time slot, the channel transitions into a new state t with probability $[[\mathbf{P}]]_{s,t}$. This transition stays independent of the packet outcome and of previous state transitions.

For our performance evaluation, we assume that the packet error probability in the good state is $p_{good} = 0$, whereas in the bad state we have $p_{bad} > 0$. We write the state transition matrix $\mathbf{P}$ as

$$\mathbf{P} = \begin{pmatrix} p_{gg} & 1 - p_{gg} \\ 1 - p_{bb} & p_{bb} \end{pmatrix} \tag{3.5}$$

where the first row corresponds to the good state and the second row to the bad state. When $0 < p_{gg} < 1$ and $0 < p_{bb} < 1$ holds, the steady-state vector

$\mathbf{\Pi}= (\Pi_0, \Pi_1)$ of $\mathbf{P}$ exists (Norris, 1997), and is given by,

$$\begin{aligned} \Pi_0 &= \frac{1 - p_{bb}}{2 - (p_{gg} + p_{bb})} \\ \Pi_1 &= \frac{1 - p_{gg}}{2 - (p_{gg} + p_{bb})} \end{aligned} \tag{3.6}$$

The average steady-state packet error rate is,

$$p = p_{good}\Pi_0 + p_{bad}\Pi_1 = p_{bad}\Pi_1 \tag{3.7}$$

The state holding times are geometrically distributed. The average state holding times (we measure them as the number of time slots) for the good state $E[H_0]$, and the average state holding time for the bad state $E[H_1]$, are given by,

$$E[H_0] = \frac{1}{1 - p_{gg}} \tag{3.8}$$

$$E[H_1] = \frac{1}{1 - p_{bb}} \tag{3.9}$$

In our experiments, we prescribe a given average channel PER (Equation 3.7), and an average duration of the bad state $E[H_1]$. We vary the so-called *burstiness index*, which is defined as,

$$B = \frac{E[H_1]}{E[H_0]} \tag{3.10}$$

For a fixed value of B, we can then solve Equations 3.6, 3.7 and 3.8 for the remaining parameters of the GE model. The burstiness index is lower-

bounded as,

$$B \geq p(2 - p_{gg} - p_{bb})/(1 - p_{bb}) \tag{3.11}$$

due to the constraint $p_{bad} \leq 1$.

3.3 Problem Formulation

We indicate the violation state of an individual stream i after the t-th superframe as,

$$V_i(t) := \begin{cases} 1 & : \quad \|\mathbf{v}_i(t)\| > k_i - m_i \\ 0 & : \quad \text{otherwise} \end{cases} \tag{3.12}$$

where $\mathbf{v}_i(t)$ denotes the sliding window state, as above. Note that $V_i(t)$ is a random variable which depends on the underlying channels and the allocated number of retransmission slots. We define the long-term average violation rate of stream i as,

$$\overline{V_i} = \lim_{t \to \infty} \frac{1}{t} \sum_{i=1}^{t} E\left[V_i(t)\right] \tag{3.13}$$

where the average is taken over all realizations of the underlying channels – the dependency on the underlying policy for allocating retransmission slots to streams is suppressed here. The overall long-term average violation rate $\overline{V}_T$, is defined as,

$$\overline{V}_T = \frac{1}{N} \sum_{k=1}^{N} \overline{V_k} \tag{3.14}$$

and this is the quantity which we aim to minimize through a proper allocation of retransmission slots.

Chapter IV

Reinforcement Learning Policy

4.1 Learning Scheduling Policy

In this chapter, we provide a detailed description of an algorithm based on Reinforcement Learning theory (Sutton & Barto, 1998). Our solution aims to minimize a discounted version of system objectives given earlier in Equations 3.13 and 3.14. The performance of the considered algorithm however, is assessed in a non-discounted way by observing the value of long-term average violation rate, $\overline{V}_T$.

4.2 Selection of a Learning Algorithm

We have chosen to employ temporal difference learning (refer to Section 2.4.4) for the learning scheme. According to the system model described in the previous chapter, a fast adaptation to changes in streams DTV is desirable. Moreover, a wireless channel error distribution would probably be unknown to an RL agent. Hence, such agent will have to initially acquire, and then use its raw experience in order to improve the scheduling efficiency.

In particular, our choice within the TD(λ_e) algorithms family focuses on the simplest TD(0)-learning. We consider the investigation of an alternative design with a general TD(λ_e)[1] as an item of future study.

Furthermore, we considered both SARSA (Sutton & Barto, 1998) and Q-Learning (Watkins & Dayan, 1992) as candidates for our learning policy. Since we could not find an apparent reason for the superiority of any one of them, we numerically compared their performance. Based on the results of this comparison (outlined in Section 7.2.1), we fixed our selection at Q-Learning.

[1] Is briefly presented in Section 2.4.4.

4.3 Introduction

Our overall approach consists of two main components. On the base station, we run one learning agent for each remote station i, which has the individual goal of minimizing a discounted version of $\overline{V_i}$ (see Equation 3.13). Mathematically, this means minimizing,

$$\overline{V_i}(\gamma) = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} \gamma^{i-1} E\left[V_i(t)\right] \tag{4.1}$$

When stream i requires a retransmission, the agent will be consulted and generates, independent of all other agents, the recommended number of retransmission slots, a_i. It may happen that the total number, $\sum_{i=1}^{M} a_i$, of requested retransmission slots (where M is the number of sensors actually needing retransmissions) exceeds the available capacity of K slots. Therefore, the other main component (referred to as coordination) comprises of an algorithm which adjusts the numbers a_i to satisfy the slot capacity constraint. It calculates revised numbers $a_i' \leq a_i$, such that $\sum_{i=1}^{M} a_i' = K$. At the end of a superframe, the outcome will be observed and will be fed back to the agents; together with the revised numbers a_i' so that the agents can learn by updating their state-action value function (also known as $Q(s, a)$).

4.4 Operation of an Individual Agent

The state of an individual agent consists of the current one-norm $\|\mathbf{v}_i\|$ of remote station i, which is being updated after every superframe (see Equation 3.3) so that the state space is $\mathcal{S}_i = \{0, 1, \ldots, k_i\}$. The action generated by an agent is the number of retransmission slots desired for stream i hence the

action space is $\mathcal{A}_i = \{0, 1, \ldots, K\}$. The agent maintains for states $s \in \mathcal{S}_i$ and actions $a \in \mathcal{A}_i$ a table of so-called Q-values, that reflect the average discounted reward of choosing action a while being in state s.

4.4.1 The Update Rule

As explained above, the coordinator might change the generated action a_i into the action $a'_i \leq a_i$. At the end of the superframe, the outcome of applying the action a'_i in state s_i is learned through observing the new state s'_i. The Q-table is then updated according to the one-step Q-learning equation (Watkins & Dayan, 1992),

$$Q(s_i, a'_i) \leftarrow Q(s_i, a'_i) + \alpha_i \cdot \left(r(s_i, a'_i) + \gamma \cdot \left(\left(\max_{a \in \mathcal{A}_i} Q(s'_i, a) \right) - Q(s_i, a'_i) \right) \right) \tag{4.2}$$

where γ denotes the discount factor and $\alpha_i \in (0, 1]$ can be interpreted as the learning rate of agent i (see below). We set the α to be small, constant and its value is identical for all agents[2]. The *max*-term[3] in parentheses is an approximation to the cumulative reward that can be earned in successor state s'_i.

4.4.2 The Rewards

The reward function $r(s, a)$ (see Equation 4.2) comprises of two parts. Its first part, denoted as $r_a(s)$, is a function of the current state s and is given

[2] we outline the method we used to find α, γ, and their values, in Chapter 7

[3] the max-term is also known as an *estimation policy* (mentioned in Section 2.4.3)

by,

$$r_a(s) := \begin{cases} r_v \times s^2 & : \quad d_i \leq 0 \\ r_d \times s^2 & : \quad 0 < d_i \leq k_i - m_i \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4.3}$$

where $r_v < 0$, $r_d < 0$ are (negative) reward coefficients. The reward $r_v s_i^2$ is assigned to agent i, in the case that stream i's deadline is violated, while the reward $r_d s_j^2$ is assigned to agent j if its stream experiences a *degradation*. A degradation can be referred to as a case where a number of the successfully received packets q_i in stream i is: $m_i \leq q_i < k_i$.

Since all rewards are non-positive ($r_a(s) \leq 0$), it follows that the rewards accumulated in $Q(s, a)$'s are non-positive as well.

Note that if the reward was only combined from the function $r_a(s)$, presented above, then under errors the agent i's only "goal" would have been to minimize stream distances to violation. We can guess that agent i (same as all others) would have been inspired, in this case, to always ask for a maximal number of slots (e.g. for K). Thereby, we define a complementary reward function that is related to agent's requested number of retransmission slots. If a_i's value turns out to be *unreasonable*, the agent i is then penalized on a positive difference between what we refer to as *modest* allocation – the $s_i \in \mathcal{A}_i$[4] and a_i, as,

$$r_b(a) := \begin{cases} r_n \left(a_i - s_i\right)^2 & : \quad a_i > s_i \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4.4}$$

where $r_n < 0$ is a constant (negative) reward intensifier. In summary, the

[4] Note that although s is a state variable, here it is used to evaluate agent's actions.

reward (denoted as r in Equation 4.2) provided to agent's Q function is,

$$r(s,a) := r_a(s) + r_b(a) \tag{4.5}$$

4.4.3 Action Selection

When the original transmission (in the uplink part) of station i fails, the agent will be asked to generate (independent of all other agents) an action a_i indicating the desired number of retransmission slots. The current state s_i is the input for this computation. The action is generated randomly from a probability distribution, based on the current table of Q-values using the softmax method[5],

$$P(a|s) = \frac{\exp(\tau(s) \cdot Q(s,a)))}{\sum_b \exp(\tau(s) \cdot Q(s,b))} \tag{4.6}$$

where the τ is dependent on stream's state. Note that for small values of τ, all the exponential terms in Equation 4.6 are close to 1, hence, $P(a|s) \approx (K+1)^{-1}$ and actions become equiprobable. A large uncertainty between different actions values, enables the agent to try different actions more often and therefore, it "explores" most of the time. As $\tau \to \infty$, high valued actions become more dominant, making the agent exploit them most of the time.

In this policy, we employed a method that allows us to efficiently maintain exploration[6]. This was originally proposed by Singh et al. (1998)[7]. Note that although its convergence proof for a softmax GLIE policy is for on-

[5] This method was first mentioned in Section 2.10 in its general form.

[6] The problem was introduced in Section 2.4.2.

[7] τ was denoted as $\beta_t(s)$ in the work of Singh et al. (1998)

policy learning algorithms, it can be generalized for Q-Learning method. The function $\tau_t(s)$ denotes $\tau(s)$ at timestep t and is defined as,

$$\tau_t(s) = \frac{\log n_t(s)}{G_t(s)} \tag{4.7}$$

where $n_t(s)$ denotes the number of visits to agent's state s in timestep t. It is assumed that the Q values are bounded. $G_t(s)$ is a balancing factor that is equal to $\max_a |Q(s, b_{max}) - Q(s, b)|$, where action $b_{max} = \arg\max_{b \in \mathcal{A}}(s, b)$[8]. In this method, the probability to select an action (in Equation 4.6) is dependent on s, t, Q and $n_t(s)$. Before we proceed with the selection of actions, we shall re-evaluate τ on every superframe and independently for each agent.

4.5 Operation of the Coordinator

It is possible, that the total number $\sum_{i=1}^{M} a_i$ of requested retransmission slots exceeds the available capacity of K slots (where M is the number of sensors requiring retransmissions in the current superframe). Therefore, the component of coordination comprises of an algorithm that adjusts the a_i's, to satisfy the slot capacity constraint.

We named our policy as N-Coordinated Q -Learning Reliability Scheduling, or briefly – NCQRS. The role of coordination in NCQRS is to balance action selection among agents and to feed this back into the process of learning. For each action a, generated by one of the individual agents, the coordinator has

[8] A detailed analysis for softmax exploration appears on page 303 (Singh et al., 1998).

to determine a corrected action a', such that the elementary constraint

$$\sum_{i=1}^{M} a'_i \leq K \tag{4.8}$$

about the total number of available retransmission slots is fulfilled. A complete schema of our algorithm is shown in Figure 4.1. $r_{a,i}$ denotes the reward

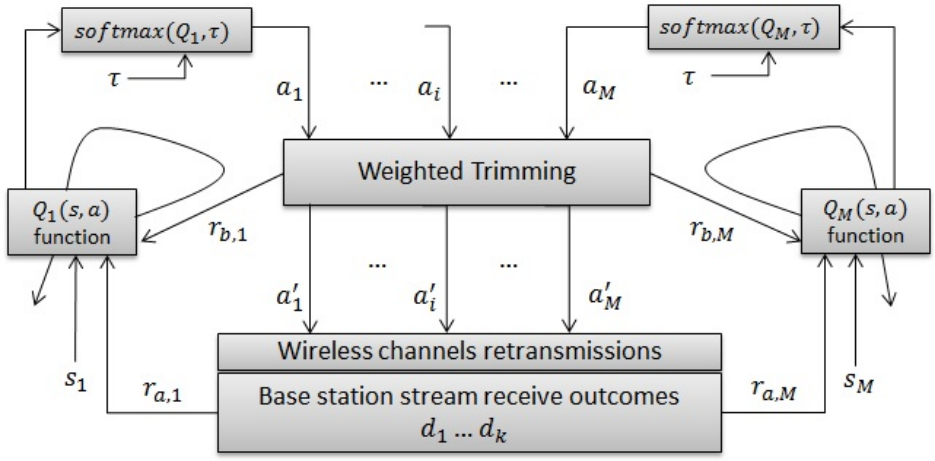


Figure 4.1: Detailed schema of NCQRS operation.

$r_a(s)$ for stream i, while $r_{b,i}$ stands for $r_b(a)$. Both the state s_i and the reward $r_{a,i}$, are determined by stream i violation state, which in turn is a result of an allocation a'_i (resulting in a'_i retransmissions) and of a wireless channel outcome. For trimming of a to a', we developed two heuristics. These can also be interpreted as social laws for agents; their description follows.

4.5.1 "Trim Best Deadline First" Algorithm (TBDF)

The TBDF algorithm proceeds iteratively, in a greedy fashion. Firstly, initialize $a'_i \leftarrow a_i$, for $i = 1 \rightarrow N$. As long as there are still excess packets (i.e. $\sum a'_i > K$), find such stream i, for which the distance d_i to the violation

of its (m_i, k_i)-firm deadline is the smallest. If multiple streams were evaluated to have identical DTV's, pick among these one stream i for which the base station has the lowest estimated packet error rate $\hat{p}_i$ (note that ties are broken randomly). Next, remove one packet from this stream and thus, do update: $a'_i \leftarrow a'_i - 1$. Note that although in this algorithm, we account for both the channel errors and the stream's distance to violation. We also give a *hard* priority to stream's DTV property over PER.

4.5.2 "Rank Based Trim" Algorithm (RBT)

In comparison to the TBDF heuristic, that favors DTV over $\hat{p}_e$, the RBT algorithm introduces *soft* (manually adjustable) preferences. Firstly, initialize $a'_i \leftarrow a_i$, for $i = 1 \rightarrow N$, and assign stream i with rank: $f_i := z_i \cdot \hat{p}_i$, where $z_i := k_i - \rho_i + 1$. Note that ρ_i is the outcome of (m_i, k_i)-firm priority evaluation described by Algorithm 4 (on page 69). Next, remove one packet from the stream with the lowest rank (denoted as j), and hence, perform an update $a'_j \leftarrow a'_j - 1$. Each time one packet is removed from a stream, its rank is multiplied by ψ. The tunable parameter ψ reduces the chance to select the already trimmed stream in the next iteration. This procedure continues until something has extinguished all excess packets, and therefore until Equation 4.8 (on page 56) is met. The parameter ψ allows us to perform a *soft adjustment* of the relative ranks and it can be adjusted to optimize the scheduling performance.

RBT Algorithm's Code

We experimented with both of the potential packet trimming heuristics and chose RBT for the generation of the final results. This algorithm produces a better performance for a selection of empirically determined values of ψ. The pseudo code of RBT algorithm is shown in Algorithm 3.

Algorithm 3 *RBT* $a, \psi \rightarrow a'$

1. **for** *streams* $i = 1 \rightarrow N$ **do**
2. $\quad a'_i \leftarrow a_i$
3. $\quad z_i \leftarrow k_i - mk\_prio(i) + 1$
4. $\quad f_i \leftarrow z_i \cdot \hat{p}_{e,i}$
5. **end for**
6. **repeat**
7. $\quad \Delta \leftarrow \sum_{m=1}^{N} a_m - K$
8. $\quad f_{min} \leftarrow \infty;\ j \leftarrow 0$
9. $\quad$ **for** $i = 1 \rightarrow N$ **do**
10. $\quad\quad$ **if** $a'_i > 0$ *and* $f_i \leq f_{min}$ **then**
11. $\quad\quad\quad f_{min} \leftarrow f_i;\ j \leftarrow i$
12. $\quad\quad$ **end if**
13. $\quad$ **end for**
14. $\quad$ **if** (j **equals** 0) **then**
15. $\quad\quad$ **return**
16. $\quad$ **end if**
17. $\quad$ **if** $\left(a'_j > 0\right)$ **then**
18. $\quad\quad a'_j \leftarrow a'_j - 1$
19. $\quad\quad \Delta \leftarrow \Delta - 1$
20. $\quad$ **end if**
21. $\quad f_j \leftarrow f_j \cdot \psi$
22. **until** $\Delta > 0$

Chapter V

Baseline Scheduling Policies

5.1 Channel Aware Optimized Scheduling (CAOS)

5.1.1 Introduction

In order to compete with NCQRS policy, we developed another baseline scheduling algorithm. It allocates retransmissions and cares *only* about the channel estimated PER. Further, we explain what kind of optimization we attempted to accomplish with regards to the allocation of packets. In this section, we formulate the optimization problem and provide a solution. We then describe practical uses of the CAOS heuristic in our work.

5.1.2 Problem Formulation

Without loss of generality, we can assume that the initial uplink transmission of the first $M \leq N$ remote stations failed. This means we need to schedule retransmissions for stations 1 to M. For each one of these stations, we want to find δ_i retransmission slots such that $\sum_{i=1}^{M} \delta_i = K$ holds and such that the probability that all M packets are successfully received after carrying out the retransmissions is maximized. This leads to the optimization problem,

$$\begin{aligned} \textbf{maximize} \quad & \prod_{i=1}^{M} \left(1 - \hat{p}_i^{\delta_i}\right) \\ \textbf{s.t.} \quad & \sum_{i=1}^{K} \delta_i = K \end{aligned} \tag{5.1}$$

where $\hat{p}_i$ $(0 \leq \hat{p}_i < 1)$ is the packet error rate estimate of stream i (see Equation 3.1). Note that each factor in the product, represents the probability that station i's packet is received when being allocated δ_i retransmission slots. To simplify notation, we shall use p instead of $\hat{p}$ in the following analysis.

5.1.3 Relaxed-Form Solution

A direct solution of the integer optimization problem, presented in Equation 5.1, is difficult because the use of integer variables makes an optimization problem *non-convex*. As an alternative, we propose to solve the logarithm of this problem in Equation 5.2. The original product transforms into a sum of logarithms,

$$\begin{aligned} \textbf{maximize} \quad & \sum_{i=1}^{M} \log\left(1 - p_i^{\delta_i}\right) \\ \textbf{s.t} \quad & \sum_{i=1}^{M} \delta_i = K \end{aligned} \tag{5.2}$$

The δ-coefficients of Equation 5.2 are integers and thus, this is still an integer optimization problem. However, if we make the assumption that for stream i, the coefficient n_i is a *relaxed version* of δ_i, we may find an approximate solution using Lagrangian relaxation[1] (Boyd & Vandenberghe, 2004),

$$F(n_1, \ldots, n_M, \lambda) = \sum_{i=1}^{M} \log\left(1 - p_i^{n_i}\right) + \lambda \left(\sum_{i=1}^{M} n_i - K \right) \tag{5.3}$$

where F is the Lagrangian and λ is the Lagrange multiplier for our constraint. We can define the M partial derivatives $\{\partial F/\partial n_1, \ldots, \partial F/\partial n_M\}$ of the Lagrangian above, and the $\partial F/\partial \lambda$. The derivative $\partial F/\partial n_i$, for some

[1] A field of mathematical optimization that allows to find an approximate solution to the original problem.

stream i, is similar to the other streams and it is,

$$\frac{\partial F}{\partial n_i} = -\left(\frac{1}{1-p_i^{n_i}}\right) \cdot p_i^{n_i} \log\left(p_i\right) + \lambda \tag{5.4}$$

$$\frac{\partial F}{\partial \lambda} = \sum_{i=1}^{M} n_i - K$$

We can find the extrema by solving $\partial F/\partial n_i = 0$ and deriving the expression for n_i,

$$p_i^{n_i} = \lambda / \left(\lambda + \log\left(p_i\right)\right) \tag{5.5}$$

assuming that $\lambda \neq -\log\left(p_i\right)$. Then,

$$n_i = \frac{1}{\log\left(p_i\right)} \log\left(\frac{\lambda}{\lambda + \log\left(p_i\right)}\right) \tag{5.6}$$

Solving $\partial F/\partial \lambda = 0$ (refer to Equation 5.4) leads to $\sum_{i=1}^{M} n_i = K$. If we substitute Equation 5.6 into that, we get,

$$\sum_{i=1}^{M} \frac{1}{\log\left(p_i\right)} \log\left(\frac{\lambda}{\lambda + \log\left(p_i\right)}\right) = K \tag{5.7}$$

We can calculate λ by solving Equation 5.7. The left hand side (LHS) of this equation is monotonically increasing inside $\lambda \in (-\infty, 0)$. Hence, it is straightforward to find the root of this equation and find λ using a numerical solver program[2]. Once λ is found, we can derive each n_i using Equation 5.6. This means that a satisfactory value for λ is one that leaves the left hand

[2] Is described further below, in Section 5.1.5.

side (LHS) of Equation 5.7 approximately equal[3] to the right hand side (i.e to K). This is addressed later in the design of solver's heuristic.

5.1.4 Allocation Coefficients

As mentioned above, when we have found λ we may use it to evaluate the allocation coefficients. However, since $n_i \in \mathbb{R}$ these values cannot be used for the allocation of packets. Thus, we denote a *rounded* version of n_i as δ'_i. The "rounding" is performed using the following rule,

$$\delta'_i := \begin{cases} \lceil n_i \rceil & : \quad n_i - \lfloor n_i \rfloor \geq \frac{1}{2} \\ \lfloor n_i \rfloor & : \quad \text{otherwise} \end{cases} \tag{5.8}$$

Rounding Error from n_i to δ'_i

We use a numeric approximation to find λ, so we expect the constraint $\sum_{i=1}^{M} n_i \approx K$ to hold. However, this may not be true anymore for the integer summation $\sum_{i=1}^{M} \delta'_i$ that may be less, or more than K.

If the condition $\sum_{i=1}^{M} \delta'_i > K$ is true, we choose to allocate δ'_1 packets to stream 1, δ'_2 packets to stream 2, and so forth until assigned all the possible K slots. Prior to this, we ensure that for each $n_i > 0$, the δ'_i is greater than or equal to 1 after the rounding.

In the case where $\sum_{i=1}^{M} \delta'_i < K$ is true (after ensuring that the rounding has not zeroed the δ''s), we randomly assign the unoccupied slots to streams until all the leftovers are distributed.

---

[3] Down to a desirable and predetermined accuracy.

5.1.5 *Numeric Solver for Finding λ*

In order to find the value of the Lagrange multiplier λ, we have chosen to apply *binary search* (Cormen, Stein, Rivest, & Leiserson, 2001). As was mentioned earlier, the function is a monotonic curve and therefore the search is simple for $\lambda \in (-\infty, 0)$. Because the binary search is known to operate within finite boundaries, we apply *exponential boundary adjustments* (Bentley & Yao, 1976) prior to it.

Figure 5.1 shows an example of a search curve for the case of two streams. The estimated channel error rates of these streams are $\hat{p}_1 = 0.1$ and $\hat{p}_2 = 0.2$.

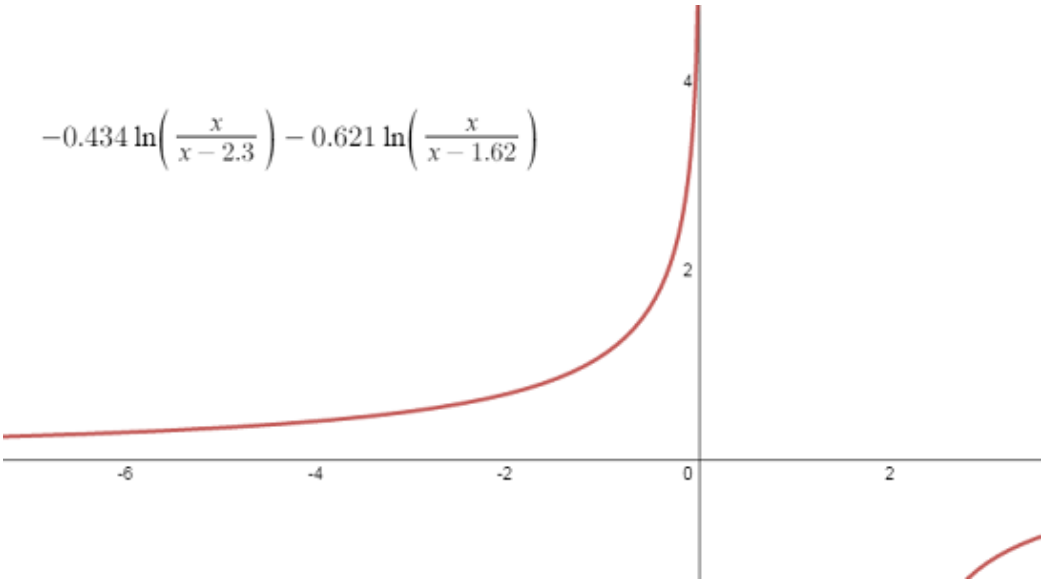


Figure 5.1: CAOS λ search curve in the example for two streams.

Firstly, the heuristic arbitrarily fixes the search boundary λ_{min}. Following this, a binary search for λ is performed within $[\lambda_{min}, 0)$. Knuth (1998) evaluated the cost of such search as $\mathcal{O}(\log n)$. If the solution is not detected

within an offered range the following exponential boundary expansion occurs: $\lambda_{min} \leftarrow 2 \cdot \lambda_{min}$. After this, a binary search is performed again over the adjusted range. This approach is followed until an approximate value of λ, given a fixed accuracy, is found. Note that a value of λ would be considered by us as a solution of Equation 5.7 in the case that $|LHS - RHS| \leq 10^{-2}$.

5.2 Distance Based Priority (DBP)

5.2.1 Introduction

And now we follow on from Chapter 2.3, where we cover the background on DBP. Here, we discuss the detailed design of our DBP-like scheduling policy. This baseline heuristic is targeted at allocating retransmissions to a population of corrupted streams and with that satisfy their (m, k)-firm deadline stream constraints. The streams with higher priorities are considered first.

5.2.2 The Method of Packets Allocation

On the base-station receiver, the DBP policy maintains N buffers, one per stream. The buffer from stream i, denoted as H_i, contains k_i cells that store binary feedbacks upon the recent k_i packets: '1' for a good packet and '0' for a packet that was corrupted. The feedback values are shifted in the buffer from the right and the rightmost element contains a feedback value of the most recent packet[4].

The policy proceeds as follows. It evaluates priorities of M streams that have reported on an error during the recent superframe. An evaluation of priorities is done in the sense of streams DTV (see Algorithm 4, page 69). The result of this evaluation is a list of priorities $\rho_1, \ldots, \rho_M$[5]. Note that the highest achievable priority for any stream is 0, while the lowest is k. Also note, that we maintain M packet allocation coefficients, denoted as δ'_1 to δ'_M, and initialize them to 0. Furthermore, for a stream that has resulted in a highest priority, the algorithm would perform an update $\delta'_i \leftarrow \delta'_i + 1$.

[4] Follows explanation of the System Model in Section 3.1.1, page 42.

[5] Stream DBP priority is defined in Equation 2.3, page 20.

The same would apply for a group of streams that have been evaluated with an identical priority; such an update would occur in random order. This policy will finish if there are no more retransmission slots available, i.e. $\sum_{i=1}^{M} \delta'_i = K$. Otherwise, the process, as described above, will be repeated until all streams priorities are equal k, i.e. the lowest priority.

We assume that the policy "flips" $0's \rightarrow 1's$ in a buffer on every stream update. This "change in buffer's state" is needed for finding the δ' coefficients. Unless flipped the buffer's priority ρ_i would remain the same for any number of iterations. Note however that although we may flip one or more $0's$ in stream's feedback buffer an actual packet replacement is feasible only for the last corrupted packet[6].

As an example, stream j has $(3, 4)$-firm deadline setting ($k_j = 4$) and the following buffer's state is $H_j = \{0, 1, 1, 0\}$. In the given state, stream j experiences violation and thus $\rho_j = 0$. On its first iteration, the policy decides to allocate one packet ($\delta'_j = 1$) and flips one '0' in the buffer, making $H_j = \{0, 1, 1, 1\}$. On the following iteration, stream j's priority would be $\rho_j = 3 < k_j = 4$. This should still trigger an allocation of another packet. The requested δ'_j for stream j is now 2, $H_j = \{1, 1, 1, 1\}$ and $\rho_j = k_j$. No more iterations are expected for stream j.

5.2.3 Limitations of the DBP Method

The DBP method presented here is sub-optimal to a setup where streams have different (m, k)-firm deadlines. For example, a priority comparison between a stream with $k = 10$ and another stream with $k = 4$, is not significant.

[6] Markov property assumption.

A possible future improvement could be to use the ideas of DBP-Matrix by Poggi et al. (2003) or E-Matrix by J. Chen et al. (2004). These techniques introduce weights in order to account for the difference in (m, k)-deadline settings of different streams. In this thesis, we assume that all streams (m, k)-firm deadlines are the same. Possible enhancements to this policy to support streams with different (m, k)-firm deadlines are a subject of future work.

5.2.4 DBP Priority Evaluation

The Algorithm 4 below, outlines the pseudo-code of a function *mk_prio*. It is based on the work of Hamdaoui and Ramanathan (1997). This function is used for the evaluation of (m, k)-firm deadline streams priorities. The function's input argument j points to a particular buffer of binary feedbacks $H[j]$. The buffer is an array of elements: $H[j][0]$ (the leftmost) up to $H[j][k-1]$ (the rightmost). The output of an *mk_prio* function is an integer from range $[0, \cdots, k]$.

Algorithm 4 $mk\_prio(j)$

```
1. if (H[j][k − 1] = 1) then
2.       return k;    # lowest priority
3. end if
4. n1s ← count 1's in H[j];
5. good_pkt ← 0;
6. if (n1s is k) then
7.       s ← 1;
8. else if (n1s < m) then
9.       s ← k + 1;
10. else
11.      s ← 1;
12.      for i = k − 1 to 0 do
13.           if (H[j][i] is 1) then
14.                good_pkt ← good_pkt + 1;
15.                if (good_pkt is m) then
16.                     s ← k − i;
17.                     break;
18.                end if
19.           end if
20.      end for
21. end if
22. return (k − s + 1);
```

Chapter VI

Simulation

6.1 OMNeT++ Simulation Framework

6.1.1 Introduction

We modeled and studied the performance of the scheduling policies described in Chapters 4 and 5 using a discrete event simulation framework called OMNeT++[1] (Varga, 2001). In addition, we designed two models of wireless channels which were discussed in Chapter 3 (from page 44).

OMNeT[2] is network simulation framework that enables its' users to rapidly develop and use a variety of "ready made" standard network protocols in the design and simulation of communication networks. OMNeT allows users to develop modules (e.g. a mobile node) in C++. It also supports a scripting language called NED which allows the user to describe networks at a high-level. For example, multiple nodes may be connected to a network using the NED language. Moreover, this simulator supports both graphical and command line interfaces, code debugging tools, tools for statistical analysis, scenario scripting tools and the functions of events logging and presentation.

6.1.2 Simulation Models and a Star Network

In our simulation, we developed four modules: the Base Station, the Remote, the Static channel and the Gilbert-Elliot channel; all written in C++. We used the NED scripting to describe a star network topology that connects one base station with a configurable number of remotes. The connections between nodes were established via one of two wireless interference models that we created.

[1] Found at www.omnetpp.org [ver 4.5].

[2] Others are NS3, OPNet, etc.

A visual representation of a network with five remotes and one base is outlined in Figure 6.1. This is one of our simulation scenarios. On the left the simulation shows a downlink transmission in progress, while on the right the simulation shows a failed transmission of one uplink packet from a bottom-left remote to the base is visualized.

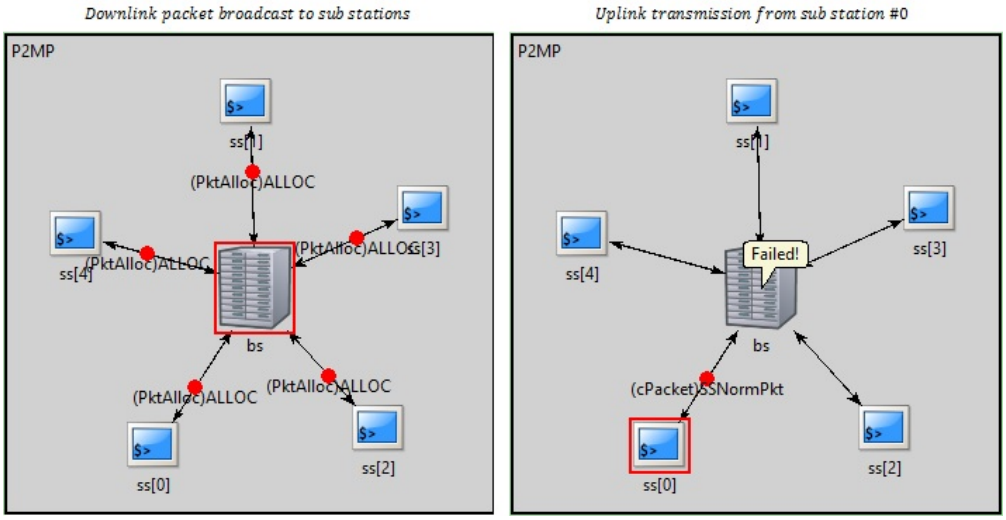


Figure 6.1: Visual simulation of five remotes and one base.

6.1.3 Experiments in OMNeT

OMNeT allows us to create and manage multiple simulation scenarios in special scripts with an *ini* extension. Essentially, any simulation parameter's value might be set via an ini-script. Moreover, one could run the same simulation scenario multiple times, for a range of parameter values. For example, we may code α = ${0.01 .. 0.02 *step* 0.001} (α is known as the learning factor), which makes the simulation repeat itself ten times: once with $\alpha = 0.01$, then with $\alpha = 0.011$, 0.012 and so forth, until α is 0.02.

6.1.4 Random Numbers Generation

We used the Mersenne Twister (Matsumoto & Nishimura, 1998) algorithm for random number generation (RNG). An example of random number usage in our code is the decision of the static channel model (page 44) to drop the received packets with probability p, drawing the values from a uniform distribution. In order to accomplish statistical confidence[3] of the results, we re-run every scenario multiple times and average all outcomes while distinguishing replications by unique seed numbers. For example, an ini-file setting *seed-set=${15,18}* would generate two replications, one with seed 15, and another one with 18.

6.2 Software Design and Architecture

6.2.1 Design Overview

The simulation includes two main modules: the base station and the remote[4]. The base station's module implements a time scheme combined from constant size *superframe* events that are generated at a fixed rate during simulation's lifetime. A group of consecutive slots defines a superframe and all the remotes are always synchronized to the base station's superframe. Finally, except for the downlink *poll packet* slot, some or all slots in a superframe are occupied by the uplink's population of remotes.

[3] This is discussed in Section 6.3.2.

[4] Sometimes, we use the term 'substation'.

6.2.2 Poll Packet

The base station generates one poll packet per superframe on the downlink. This packet contains feedback of the scheduler's decisions from the previous superframe – a list of 2-tuples such as $\langle id, \delta' \rangle$, where $id \in \{0, 1, \ldots, N-1\}$ is a remote's identifier, and $\delta' \in \{1, 2, \ldots, K\}$ is the number of allocated retransmissions for that id. For example, an advertised list of $\langle\langle 3, 2\rangle\langle 4, 1\rangle\langle 5, 2\rangle\rangle$, suggests that remotes with ids 3 and 5, would have to send two repetitions of their previous data packets, while id 4 would need to resend once.

6.2.3 Remote's Traffic

In this work, we assume a traffic model in which the poll always triggers one transmission of a user data packet from every remote. In addition to these packets, and following the 2-tuples base station's list, some remotes would also generate one or more retransmissions. An assignment of transmission slots at the remote is covered in detail within Section 3.1.2, on page 43.

6.2.4 Wireless Interference Models

The channel models are implemented as logic within the base and remote station receiver's code. The useful outcomes are decisions on every received packet of whether to *mark* it as corrupted or not. The underlying logic is implemented inside a base and remote for both channel models (see Section 3.2, page 44). Note that due to an error-free downlink conjecture, packets are always considered *correct* at the remote's end.

6.2.5 *Support of (m, k)-Firm Streams*

The base station model implements the support for a stream's (m, k)-firm deadlines metric. It provides the base with an evaluation of the DTV and priority[5] for each stream. It also creates and maintains N binary feedback buffers $H_1, \ldots, H_N$, as discussed in Section 5.2.2, on page 66.

6.2.6 *Base Station State Machine*

The process of our base station model can be described by a state machine, denoted as BS-SM. Its diagram is shown in Figure 6.2.

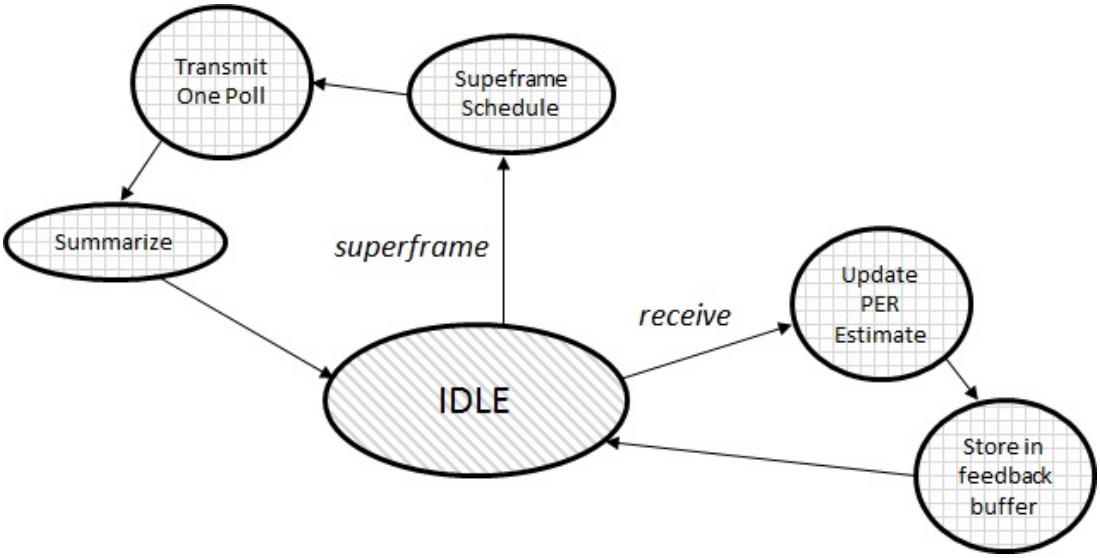


Figure 6.2: Base station State-Machine Diagram.

From an *IDLE* state, BS-SM may respond to the recurring superframe event. Following this, a scheduler's procedure is executed (the current scheduling policy is a parameter in simulation). The scheduling is followed by the

[5] The DTV and priority ρ are defined in Equations 3.4 and 2.3, correspondingly.

transmission of a poll packet. Note that we assume the scheduler's computation time is negligible.

Additionally, the base station evaluates the system's state in terms of the violation of (m, k)-firm deadlines, defined as the average of an immediate violation for stream i on a t-th superframe (see Equation 3.12, page 48), for all the remotes.

Furthermore[6], a new feedback is recorded ('1' or '0') on an uplink packet *receive event* from remote i. This feedback is then pushed into buffer H_i from the right[7]. It is also used to update the PER estimate (Equation 3.1, page 42) for substation i.

6.2.7 System Violation Rate

In this work, we compared the performance of studied policies in terms of how well they manage to avoid (m, k)-firm deadlines violations. A value of $V_i(t)$ (Equation 3.12, page 48) gets '1' for violation, or '0', otherwise. Therefore, this function is a sequence of successes and failures. A system violation rate for the t-th superframe is the average: $V_T(t) = \frac{1}{N} \sum_{i=1}^{N} V_i(t)$. Further, we shape the function $V_T(t)$ by applying a moving average[8] to it, and denote the resulting function as $\overline{V}_T(t)$.

An example of functions $V_T(t)$ and $\overline{V}_T(t)$, is shown in Figure 6.3. The average is evaluated for three streams with arbitrarily selected values:

$$t_1 = \{1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0\}$$

[6] Also from an *IDLE* state.

[7] Defined in Section 5.2.2; buffers work like FIFO (First In, First Out).

[8] Similar to what we did in Equation 3.1, page 42.

$$t_2 = \{1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0\}$$
$$t_3 = \{1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0\}$$

over twelve superframes and with a smooth factor of 0.1. Note, that if a simulation replication j has run for long enough to discard all transient effects, then the mean of our observations, $\overline{V}_T^{(j)}$, should represent the system's long-term average violation rate for policy in steady state.

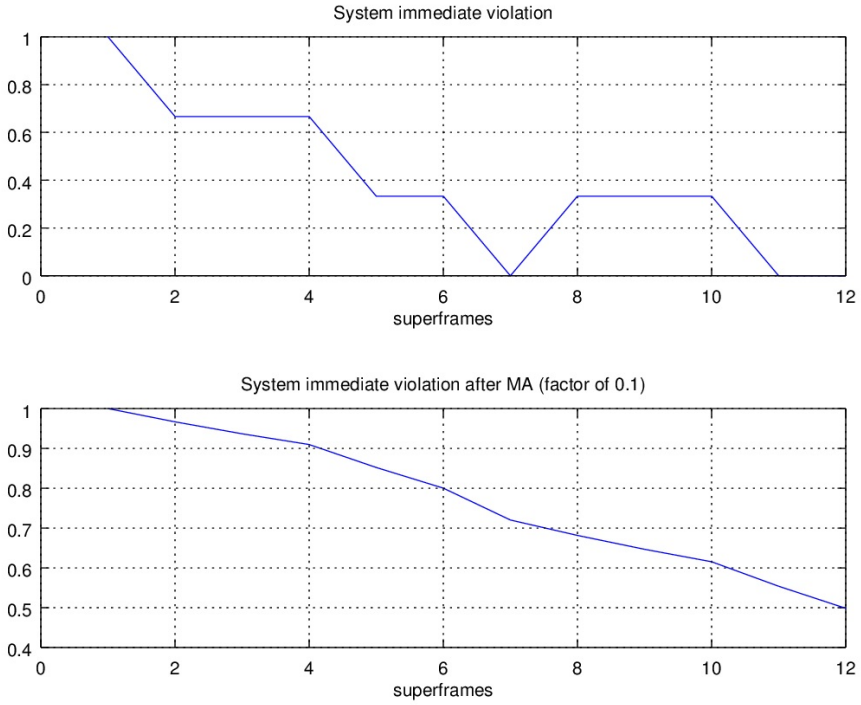


Figure 6.3: System violation rate before and after a moving average.

6.2.8 Simulation Events, Slots and Duration

Our simulation can generate a large number of events. For example, a poll packet transmitted to N remotes in broadcast, is composed of N individual

OMNeT events: first to remote 1, second to remote 2, etc. These events still occur within a single simulation time slot, which we shortly denote as slot. The duration of one superframe is known as $N + K$ slots. We consider the simulation time in terms of a number of slots. For example, if one superframe contains twenty slots and we allow time for 30,000 superframes, the simulation's time would then be 500,000 slots. Note that we set a duration long enough to assure that the simulation resides most of the time in a steady-state.

6.3 Simulation – Experimentation and Analysis

We can identify our experiments as three independent groups. The first deals with a numeric search of optimal parameters that are later used by the learning policy. The second group of experiments is related to a comparison of performance of the studied policies, i.e. the CAOS, DBP and NCQRS. The final group deals with the study of convergence times for NCQRS. We denote these groups as G_A, G_B and G_C, respectively.

6.3.1 Fixed Parameters

We fixed simulation time to $500,000$ [slots] for all scenarios except for the group of convergence tests (G_C), where it was extended to 10^6 [slots].

In this work, we examined scenarios with 5, 10, 15 and 20 remotes. A setup with a larger number of nodes is considered as a future study item. Given a number of remotes, we determine the size of one superframe as follows: we assume a fixed ratio of 2 retransmission slots per remote, e.g. for 5 nodes, the number of retransmission slots is 10 and the total supeframe

size in this case is 15 [slots]. Note that in practice, the scheduling policy determines how many slots will each remote occupy at a given supeframe out of the total number of retransmission slots.

Wireless Channels Parameters

As mentioned before, we use $\mu = 0.1$ for channel packet error rate estimation (with Equation 3.1); this applies to all links. The base station keeps the array of the estimated values $\hat{p}_1, \ldots, \hat{p}_N$ arbitrarily initialized to 0.1. Each scenario also conveys a set of real channel error rates, that we denote as $p_1, \ldots, p_N$. These are used by our wireless channel models for their decision on the marking of packets as corrupted.

In the Gilbert-Elliot channel model's logic, we fix $e_G = 0$ and convey values for $e_B, E[H_1]$ and B[9]. For this simulation, we initially set $e_B = 1$. This value is then adjusted (before run) so that the channel's mean PER is approximately p. The adjustment process is as following: we know from Equation 3.7 that $e_B = p/\Pi_1$, where Π_1 can be found from Equation 3.6. By evaluation of probabilities p_{bb} and p_{gg} we find that p_{bb} could be directly evaluated as $1 - (E[H_1])^{-1}$; $p_{gg} = 1 - (E[H_0])^{-1}$, where $E[H_0] = E[H_1]/B$, which follows from Equation 3.10.

In some scenarios, a subset or all of the above mentioned parameters might be different for each remote's uplink channel. For example, in a fully heterogeneous Gilbert-Elliot channel scenario, the values of $p, E[H_1], B$ and e_B, would be unique for each channel.

[9] The channel model is explained in detail in Section 3.2, page 44.

Reinforcement Learning Policy Parameters

On the base station, we configure the NCQRS policy learning coefficients α and γ, reward factors r_v, r_d, r_n and the RBT heuristic parameter ψ. These parameters are set with *optimal* values discovered in experiments that belong to group G_A[10] (see the results chapter, Sections 7.2.2 and 7.2.3). The mentioned parameters are common to all the RL agents.

We examined the 5-tuple: $\langle \alpha, \gamma, r_v, r_d, r_n \rangle$. If n denotes the number of replications, then a brute-force search will require $n \cdot k_\alpha \cdot k_\gamma \cdot k_v \cdot k_d \cdot k_n$ iterations which is impractical. Note that the values $k_\alpha, k_\gamma, k_v, k_d$ and k_n, denote the number of levels used for each element of the 5-tuple, correspondingly. In order to reduce search size, we assumed the factors α and γ are independent of the rewards[11]. The search was split into two steps. First, we varied α and γ with fixed values for r_v, r_d and r_n. Then we used the optimal values of α and γ found in step one and varied the rewards. Together with this, we made the assumption for r_d to stay fixed and varied r_v and r_n.

The scenarios over which we optimized the learning policy parameters are described in Sections 7.2.2, for Static channels, and in 7.2.3, for Gilbert-Elliot channels.

Parameters for (m, k)-Firm Deadlines

In this thesis, we chose configurations such as $(3, 4)$-firm and $(8, 10)$-firm, for the experimental group G_B. For the experiments in G_A, we only used the $(3, 4)$-firm deadline setting, while in some of the convergence tests of group

[10] We acquired $\psi = 1.2$, but do not present the preliminary work we did to find it.

[11] This assumption was driven by past observations.

G_C, a wider range of configurations was employed: $(4, 6)$, $(6, 8)$, $(8, 10)$, $(5, 10)$, $(10, 12)$, $(12, 14)$. Note that we assigned identical (m, k)-firm deadlines to all streams. Scenarios in which the firm (m, k)-firm deadlines differ for some streams are considered as a future study item.

6.3.2 Statistical Significance of Simulation Results

In order to attain results that are *statistically significant*, we employed the method of *replications*. A replication must be based on a different, stochastic and independent sequence of numbers (Law & Kelton, 2000; Pawlikowski, 1990). If we employ an OMNeT technique with a unique seed (integer number) for each replication (earlier mentioned in Section 6.1.3, page 72) we can create such independence.

Furthermore, a sample size (i.e. a number of replications), denoted as n, must be determined before the actual experiment. Later in this Section, we will show how the sample size can be evaluated.

Earlier, in Section 6.2.7 on page 76, we presented the violation rate function $\overline{V}_T(t)$. We compute the mean for observations gathered in replication j. This mean is denoted as $\overline{V}_T^{(j)} \in [0, 1]$. In summary, given a sample of size n, our violation rate result is,

$$\overline{V}_T := \frac{1}{n} \sum_{i=1}^{n} \overline{V}_T^{(i)} \tag{6.1}$$

where $\overline{V}_T$ can be treated as a statistically significant result.

Sample Size Evaluation

We are required to find the sample size prior to each new experiment (or simulation scenario). For that purpose, we used the following technique: we perform a *large* number of replications, denoted as n_∞. Our "prediction"[12] is: $n_\infty \geq n$. We performed a procedure that tests this prediction; it is described in the following paragraph. In the case that our prediction was false, we would empirically increase the value of n_∞ and repeat the procedure. Based on this, we retest our hypothesis until its true and n could be determined.

Let us denote the k-th sample mean $\overline{x}_k$ and sample variance U_k^2, where $k \in 1, \ldots, n_\infty$ as,

$$\overline{x}_k := \frac{1}{k}\left(\overline{V}_T^{(1)} + \cdots + \overline{V}_T^{(k)}\right) \tag{6.2}$$

$$U_k^2 := \frac{1}{k}\sum_{i=1}^{k}\left(\overline{V}_T^{(i)} - \overline{x}_k\right)^2 \tag{6.3}$$

We assume a confidence level of 95% and write the half-width of the confidence interval as,

$$\Theta_k := t_{\frac{\alpha}{2},k-1} \times \frac{U_k}{\sqrt{k}} \tag{6.4}$$

where $\alpha = 0.05$ and $t_{\frac{\alpha}{2},k-1}$ is a t-student distribution with $k-1$ degrees of freedom (Jain, 1991). Because of a sufficiently large sample sizes in our simulation, we were to replace the t-distribution by standard normal distribution, hence, $z_{\alpha/2}$ – the $\frac{\alpha}{2}$-quantile of the standard normal distribution, may take over $t_{\frac{\alpha}{2},k-1}$. The confidence interval of a measurement can be defined as,

$$c_k := (\overline{x} - \Theta_k, \quad \overline{x} + \Theta_k) \tag{6.5}$$

[12] The minimal size of the required sample is unknown at this stage.

Furthermore, we plot the set of each mean $\{\overline{x}_1, \ldots, \overline{x}_\infty\}$, the sample variance set $\{U_1^2, \ldots, U_\infty^2\}$ and the confidence interval relative to the sample mean $\{2 \cdot \Theta_1/\overline{x}_1, \ldots, 2 \cdot \Theta_\infty/\overline{x}_\infty\}$. We may determine n by visual inspection of the above plots. The sample size is defined as the smallest number of replications, for which the last quantity is lower than 1%. We also ensure that U_k^2 becomes small as $k \to n_\infty$ since the generation of stable and short confidence intervals depends on it (Pawlikowski, 1990).

6.3.3 Convergence Speed of NCQRS Policy

Here, we outline an additional detail of simulation that we introduced to measure convergence speed of our learning policy. We do this under a variety of conditions[13]. An instantaneous change in the packet error rate of all uplinks is introduced. This triggers the agent's adaptation to new conditions. We define the PER step function $p_i(t)$ for channel i as,

$$p_i(t) := \begin{cases} p_{s,i} & : \quad t < t_s \\ p_{e,i} & : \quad t \geq t_s \end{cases} \tag{6.6}$$

where $p_{s,i}$ and $p_{e,i}$ are constants and $p_{e,i} > p_{s,i}$. We set both arbitrarily, e.g. $p_{s,i} = 0.2$ and $p_{e,i} = 0.3$. We may define $p_h := (p_{e,i} - p_{s,i})$ – a positive increase in PER for channel i.

Initially, we increased p_h and studied how this affects convergence[14]. Additionally, while assuming one fixed p_h, we trialled systems each time setting different (m, k)-firm deadlines. Our goal was to test if an expansion of every

[13] Experiments on speed of convergence were classified under group G_C; the results are presented in Section 7.5 on page 101.

[14] Scenarios were homogeneous; each uplink is trialled with an identical PER change.

agent's state space $\mathcal{S}$ (because k goes up) would introduce a notable increase in convergence time.

We repeated the experiments for systems of 10, 15 and 20 users. The measured time was evaluated in superframes (not in slots), in order to achieve a consistent comparison between systems with different numbers of users, i.e systems with different superframe lengths.

Measurement of Convergence

We limited the study of convergence to the Static channel model; the Gilbert-Elliot model may in some cases introduce a non-stationary behavior. Study of learning policy convergence over channels with time-varying characteristics is an item of future study.

As mentioned above, we measured the convergence time of policy "response" to a *significant* PER step-change. The policy performance results in an increase of violation rate. We would expect the response to begin after t_s but finish before t_e, as shown in Figure 6.4. Additionally, we define t_c as a point where the policy convergence is assumed to be completed, i.e. we anticipate that our policy is in steady state after t_c.

In order to find the length of this transient (evaluated as $\Delta t_c := t_c - t_s$) we employed the nonoverlapping batch means method[15] (Law & Kelton, 2000; Pawlikowski, 1990) in this work. We set a constant value for t_s. Then, we determine the time instant t_c as follows: the mean of replication j is known as $\overline{V}_T^{(j)}$. After convergence this should be approximately equal to the value of few final batches of observations. We denote the fixed batch length as L

[15] Compared to overlapping batch means, this method is less computationally complex.

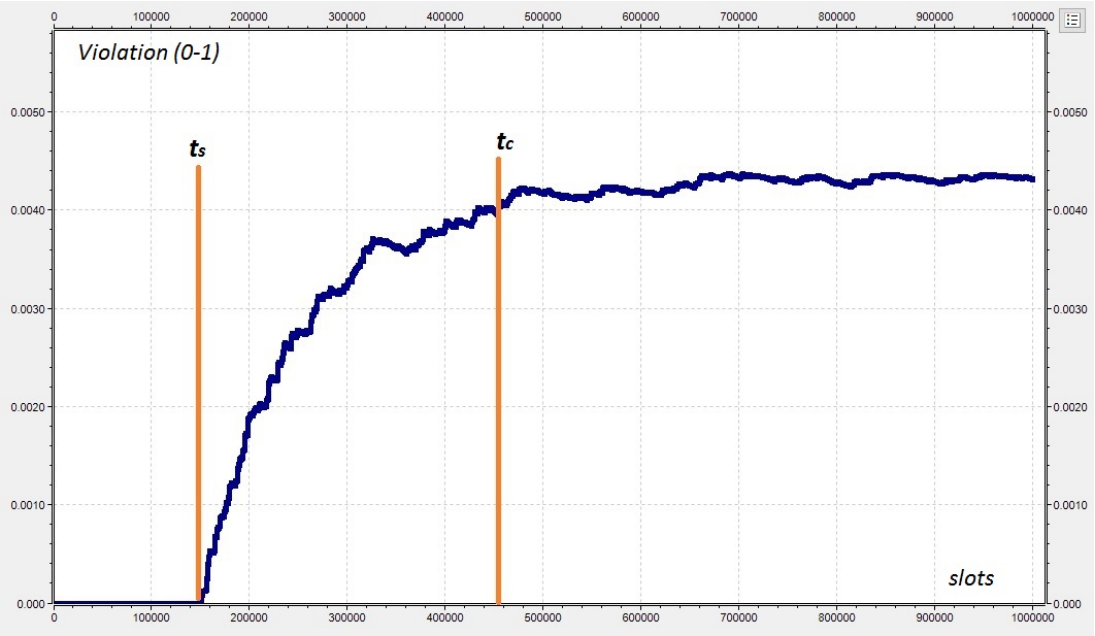


Figure 6.4: NCQRS Response to change in PER – 15 users $(5, 10)$-firm system; $p_s = 0.2$; $p_e = 0.45$, one replication.

(in superframes). Next, the data of replication j is divided into $q := t_s/(l \cdot L)$ batches (l is a superframe duration in slots), and the mean of each batch is evaluated. Let us denote the mean of batch q as $\overline{V}^{(j)}_{T,b_q}$. Starting from b_1, we perform the comparison,

$$\omega \overline{V}^{(j)}_{T} \leq \overline{V}^{(j)}_{T,b_i} \tag{6.7}$$

where ω is some threshold ($\omega < 1$) and the batch index i may run up to batch q. However, we terminate the search the first time Equation 6.7 becomes true. The outcome of replication j is some batch index $1 < q' \leq q$ that allows us to evaluate the "stop time" t^j_c as,

$$t^j_c := L \cdot (q' - 1/2) \cdot l \ [slots] \tag{6.8}$$

The procedure above is repeated for n replications (the process of finding n is described in Section 6.3.2). Eventually, the convergence time of an experiment can be written as the average,

$$\Delta t_c := \frac{1}{n} \sum_{i=1}^{n} \Delta t_c^i \tag{6.9}$$

where $\Delta t_c^j := t_c^j - t_s^j$ denotes the convergence time found from replication j.

6.4 Verification

We tested the simulation code validity, first for its individual modules and then as a complete system. In this section, we discuss the validation of individual simulation modules.

6.4.1 Channel Models

For the static channel model, we set different arbitrary values for the channel's PER and observed the estimated value $\hat{p}$, as function of time. The behavior of function $\hat{p}(t)$ matched our expectations with respect to the exponentially weighted moving average estimator in Equation 3.1. Additionally, we set different values of μ and observed estimation performance changes in $\hat{p}(t)$.

For the Gilbert-Elliot channel model, we studied multiple $\hat{p}(t)$ plots while changing the values of B, $E[H_1]$ and e_B (we kept $e_G = 0$). The behavior of function $\hat{p}(t)$ was consistent with findings by Willig (2005), who used an identical channel model.

6.4.2 Distance Based Priority and (m, k)-Firm Buffers

In order to verify the evaluation of priority ρ_i within the receive buffer of stream i, we injected *known* sequences of 1's and 0's into that buffer. The evaluation of distance d_i and priority ρ_i, done by Algorithm 4 (page 69), was consistent with the original work of Hamdaoui and Ramanathan (1995). Additionally, we experimented with different (m, k)-firm deadline settings using the same approach.

6.4.3 Stream States, Rewards and Violation Rate

Based on the assumption we correctly evaluate the DTV, we were able to confirm the expected stream's state and its corresponding immediate reward, according to Equation 4.3. Moreover, it was not difficult to manually evaluate the violation rate V_i of stream i for relatively short input binary sequences.

6.4.4 CAOS Policy

The CAOS policy uses Equations 5.6 and 5.7 to evaluate allocation coefficients. We ran this policy for a set of predetermined inputs. For comparison, we used a spreadsheet of CAOS equations and identical inputs. The spreadsheet was designed to confirm the heuristic's behavior. Another proof of CAOS correct operation was the constraint test $\sum_{i=1}^{M} n_i \approx K$, where $n_i \in \mathbb{R}$ are the evaluated allocation coefficients and K denotes the total number of available retransmissions slots.

6.4.5 NCQRS Policy

The verification of our learning policy was done over indirect outcomes such as the violation rate (a system measure of performance). We manually varied α and γ and observed the algorithm's operation. In some cases, NCQRS worked well, bringing the violation below 1%. However, for some other values of learning coefficients, it diverged and the $\overline{V}_T$ reached 30% or more. We also varied other heuristic parameters like ψ and the reward coefficients. We examined both the average policy performance, i.e. the $\overline{V}_T$ and also the individual violation rate of each link.

Chapter VII

Results

7.1 Introduction

In this chapter, we discuss the experiments and their results. Table 7.1 below presents a summary of simulation experimental groups initially defined in Chapter 6 on page 78. The NCQRS heuristic uses Q-Learning[1] for its experience update rule. For multiple scenarios that were categorized into groups G_A, G_B and G_C, we specify the number of remotes, wireless channel types and quantities, that we varied.

Group	Sub-stations	Channel type	Variables
G_A	5	*static* p_{hom}, p_{het}	$\alpha, \gamma, r_v, r_d, r_n$
G_A	5	*G.E.* p_{hom}	$\alpha, \gamma, r_v, r_d, r_n$
G_B	5, 20	*static* p_{hom}, p_{het}	p, m, k
G_B	5, 10, 15, 20	*G.E.* p_{hom}, p_{het}	$p, B, E[H_1]$
G_B	10, 15, 20	*static* p_{hom}	p, m, k
G_C	10, 15, 20	*static* p_{hom}	p_s, p_e, m, k

Table 7.1: General summary of the experimented configurations.

Previously, we explained the assumptions used in our system's setup and provided values for various simulation parameters in experiments, defined by Table 7.1. Some were mentioned in Section 6.3, while we addressed others in the design of policies – Chapters 4 and 5.

7.2 Group G_A – Parameter Space Optimization

The optimization of learning policy parameters was of our concern prior to performance comparison (in G_B) or convergence (in G_C). We were focused on four sample scenarios, detailed in Table 7.2. Note that we limited ourselves

[1] The preference of Q-Learning over SARSA was based on performance comparison outlined in Section 7.2.1.

to a low number of experiments because every search is a lengthy computational exercise. Nonetheless, based on past exploratory simulation work that we did (but not presented in the thesis), we are confident that these optimal parameters would not differ significantly for systems with more than 5 and up to 20 substations. We consider the process used for optimization as burdensome – it involved multiple manual actions, e.g. visual inspections and parameters adjustments.

Experiment	$p_1, \ldots, p_5$	$E[H_1]/B$
Static p_{hom}	0.2 0.2 0.2 0.2 0.2	-
Static p_{het}	0.1 0.15 0.2 0.25 0.3	-
G.E. p_{hom}	0.2 0.2 0.2 0.2 0.2	50/0.5
G.E. p_{het}	0.1 0.15 0.2 0.25 0.3	50/0.5

Table 7.2: NCQRS parameter optimization scenarios.

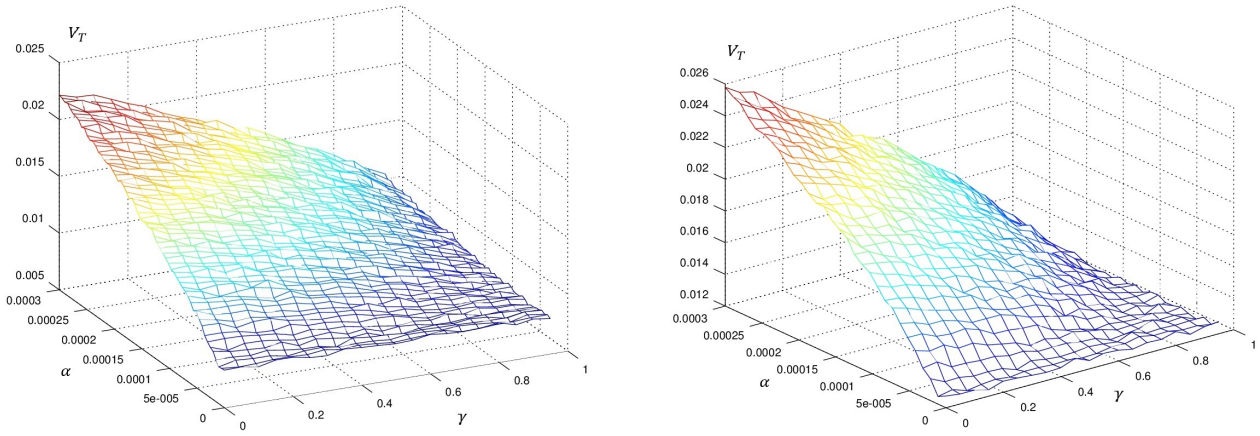
Given the scenarios in Table 7.2, we looked at $\overline{V}_T$ of our algorithms within a parameter space that is defined by the 5-tuple $\langle \alpha, \gamma, r_v, r_d, r_n \rangle$. The strategy of finding optimal values for these policy parameters is discussed in Section 6.3.1 on page 80.

7.2.1 SARSA or Q-Learning

We compared the performance of Q-Learning and SARSA techniques, in order to pick up a candidate for our RL policy as follows. First, we did the experiments of Group G_A for both SARSA and Q-learning. Then, we looked at their performance in terms of $\overline{V}_T$ within the examined parameter space. Based on the results, Q-Learning outperformed SARSA.

We show the comparison of one scenario: the static p_{hom} (Table 7.2).

Figure 7.1a shows a surface plot of $\overline{V}_T(\alpha, \gamma)$ for Q-Learning and Figure 7.1b is for SARSA. This figures show that, Q-Learning managed to reduce violation twice better than SARSA.



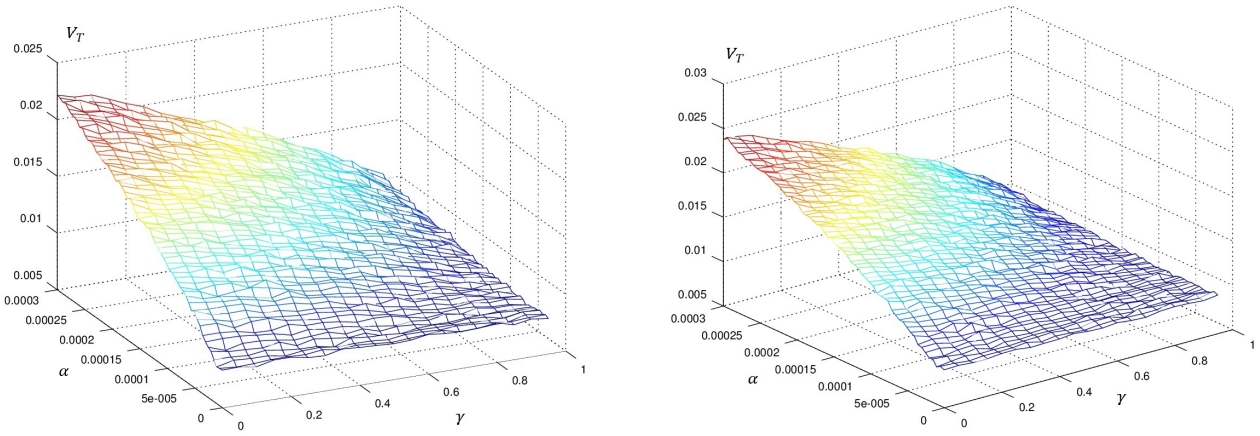
(a) (b)

Figure 7.1: Function of $\overline{V}(\alpha, \gamma)$ for Q-Learning and SARSA over Static homogeneous channels.

7.2.2 Optimal Parameters Search over Static Channels

Following the optimization method that was discussed in Section 6.3.1, we carried out static channel experiments from Table 7.2. Our initials assumptions were: $r_n = -0.02, r_d = -10$ and $r_v = -100$. We varied $\gamma \in \{0, \ldots, 0.95\}$ with steps of 0.05 and $\alpha \in \{5 \times 10^{-7}, \ldots, 3 \times 10^{-4}\}$ with step of 5×10^{-6}. The sample size was $n = 90$. Both of the $\overline{V}_T$ functions (surface plots) are shown in Figure 7.2a for Static homogeneous scenario, and in Figure 7.2b for the heterogeneous one. By visual inspection, we determined the optimal values of the first step: $\alpha \in \{0.00005, \ldots, 0.00015\}$[2] and $\gamma = 0.95$.

[2] Within the given range, all α's were equally optimal.



(a) (b)

Figure 7.2: $\overline{V}_T(\alpha, \gamma)$ over Static p_{hom} and p_{het} channels.

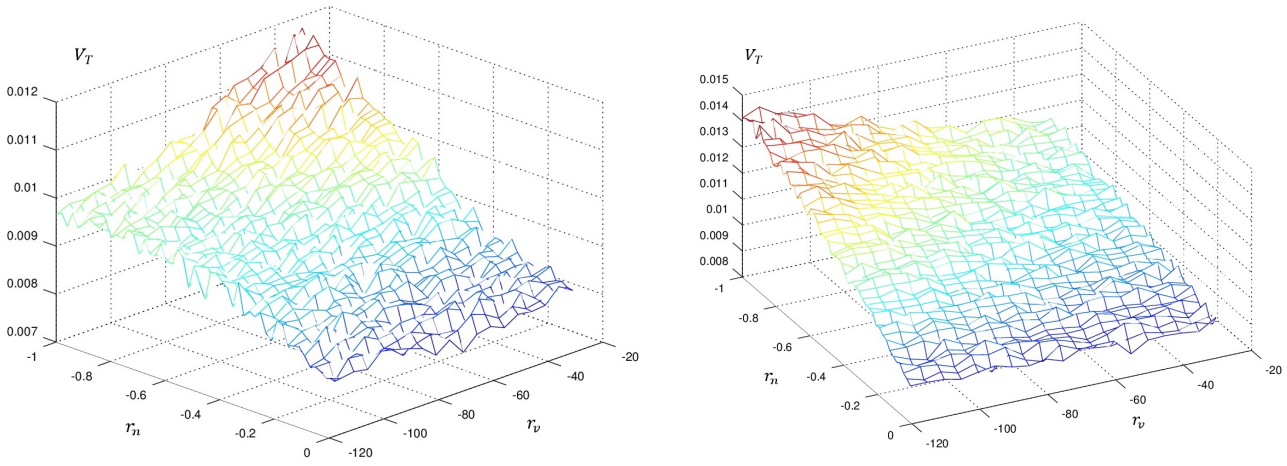
Further, we proceeded with finding optimal reward coefficients, given $\alpha = 0.0001$ and $\gamma = 0.95$. We fixed $r_d = -10$ and drew r_n from $\{-0.01, \ldots, -1\}$ with step of -0.02 and r_v from $\{-120, \ldots, -30\}$ with steps of 5; 90 replications were used. The results of this experiment are outlined in Figures 7.3a and 7.3b for Static homogeneous and heterogeneous channels, respectively.

After the above process, the optimal parameters for learning policy in communication over static wireless channels, both homogeneous and heterogeneous, are,

$$\alpha = 10^{-4}, \gamma = 0.95, r_d = -10, r_v = -70, r_n = -0.02. \tag{7.1}$$

7.2.3 Optimal Parameters Search over G.E. Channels

We replayed the previous experiment with a similar system setup, but used Gilbert-Elliot channels. Both the homogeneous and heterogeneous (in terms



(a) (b)

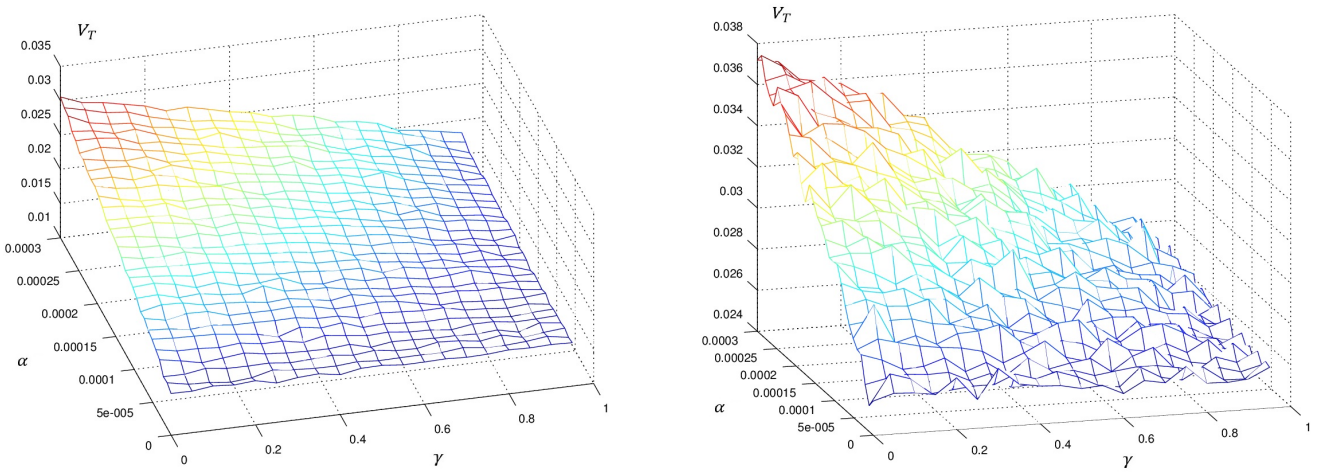
Figure 7.3: $\overline{V}_T(r_n, r_v)$ over Static p_{hom} and p_{het} channels.

of PER only) scenarios were explored (see Table 7.1 for Gilbert-Elliot configurations). In both experiments, we had to do 150 replications ($n = 150$). Similar to optimization over Static channels, we outline the functions $\overline{V}_T(\alpha, \gamma)$ in Figures 7.4a and 7.4b, for G.E. p_{hom} and p_{het} channel scenarios respectively.

Study of the results gave us $\alpha = 0.0001$ and $\gamma = 0.95$. We then used these for finding reward coefficients. Moreover, we were able to find optimal and common coefficients for learning between Static and Gilbert-Elliot channel experiments. Figures 7.5a (p_{hom}) and 7.5b (p_{het}) show surface plots of $\overline{V}_T$ as function of r_n and r_v reward coefficients. Note, that $r_d = -10$ and the number of replications is $n = 150$.

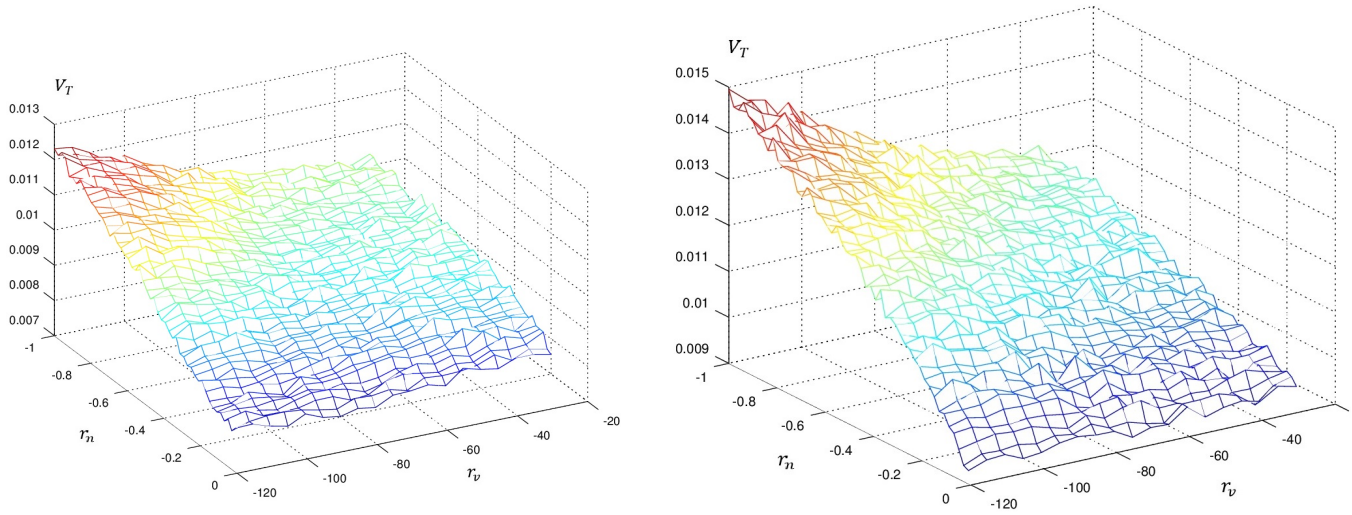
In summary, for both experiments we found the optimal 5-tuple based on our search method is,

$$\alpha = 10^{-4}, \gamma = 0.95, r_d = -10, r_v = -70, r_n = -0.02. \tag{7.2}$$



(a) (b)

Figure 7.4: $\overline{V}_T(\alpha, \gamma)$ over G.E. p_{hom} and p_{het} channels.



(a) (b)

Figure 7.5: $\overline{V}_T(r_n, r_v)$ over G.E. p_{hom} and p_{het} channels.

7.3 Performance Comparisons over Static Channels

In this experiment, we investigated the performance of 5 and 20 remotes over Static wireless links. We observed levels of $\overline{V}_T$ over both the p_{hom} and p_{het} channel scenarios at five different error rates $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for a 5-user system and at another five error rates $\{0.1, 0.15, 0.2, 0.25, 0.3\}$ for a 20-user system[3]. The number of replications for each error rate was 100.

Note that if we have a 5 user homogeneous scenario in a point $p = 0.2$, this would mean that $\overline{p} = p_1 = \cdots = p_5 = 0.2$. Conversely, for heterogeneous channels, each uplink may be assigned with a different value. Table 7.3 specifies heterogeneous scenario channels error rates for system of 5 users, while Table 7.4 does that for 20 users system.

Nodes	$\overline{p} = 0.1$	$\overline{p} = 0.2$	$\overline{p} = 0.3$	$\overline{p} = 0.4$	$\overline{p} = 0.5$
1	0.03	0.06	0.13	0.22	0.39
2	0.14	0.25	0.28	0.35	0.21
3	0.20	0.2	0.43	0.45	0.52
4	0.08	0.15	0.11	0.13	0.37
5	0.05	0.09	0.05	0.1	0.09

Table 7.3: Packet error rates for 5-user system over Static heterogeneous channels.

Figures 7.6 (page 104) and 7.7 (page 105) outline the long-term average violation rate ($\overline{V}_T$) of the three studied policies as a function of packet error rate. These experiments were repeated for the $(3, 4)$-firm and for a more *relaxed* $(8, 10)$-firm deadline setting. The $\overline{V}_T$ ranged from 0 to 1. For example, a value of 0.08 means that streams have spent 8% of their time (on

[3] In NCQRS, 20-user systems diverge above $\overline{p} = 0.35$.

Nodes	$\overline{p} = 0.1$	$\overline{p} = 0.15$	$\overline{p} = 0.2$	$\overline{p} = 0.25$	$\overline{p} = 0.3$
$1, 6, 11, 16$	0.03	0.13	0.39	0.57	0.4
$2, 7, 12, 17$	0.14	0.28	0.21	0.33	0.6
$3, 8, 13, 18$	0.20	0.43	0.52	0.59	0.3
$4, 9, 14, 19$	0.08	0.11	0.27	0.37	0.7
$5, 10, 15, 20$	0.05	0.05	0.09	0.16	0.5

Table 7.4: Packet error rates for 20-user system over Static heterogeneous channels.

average) in violation. NCQRS showed the lowest $\overline{V}_T$ in a setup of 5 users for $\overline{p} \leq 0.2$, however, and was outperformed by DBP and CAOS in a 20-user system. Additionally, we learned that the learning policy has a limited operational range, especially for cases of high $\overline{p}$. Although it is important to understand how the heuristic works over Static channels, this has little practical relevance, because this channel model is highly idealized.

7.4 Performance Comparisons over Markovian Channels

In this part, we considered scenarios with 5, 10, 15 and 20 substations (see Group G_B in Table 7.1). We investigated the performance of NCQRS, DBP and CAOS policies in terms of $\overline{V}_T$, for a fully homogeneous scenario and over different heterogeneous scenarios[4]. Additionally, for Gilbert-Elliot channels, we only used the $(3, 4)$-firm deadline setting.

[4] In a fully heterogeneous scenario, the $p, E[H_1]$ & B are different for each link; in homogeneous scenarios, all links have identical properties.

7.4.1 $\overline{V}_T(\overline{p})$ with Fixed B in Heterogeneous Scenarios

Here we compared the long-term average violation ratio between policies, while the channels error rates were increasing. Initially, we fixed $B = 0.5$ and considered two independent scenarios. The first setup had 10 users, for which unique values for $E[H_1]$ and PER were set (refer to Table 7.5). The second 20-users setup was homogeneous in terms of PER; we assigned different $E[H_1]$ values for streams: $\{2, 4, 6, \ldots, 36, 38, 40\}$ to streams $1, \ldots, 20$, respectively.

Nodes	$\overline{p} = 0.1$	$\overline{p} = 0.15$	$\overline{p} = 0.2$	$\overline{p} = 0.25$	$\overline{p} = 0.3$	$E[H_1]$
1	0.03	0.08	0.06	0.31	0.33	5
2	0.05	0.14	0.07	0.195	0.31	10
3	0.13	0.13	0.15	0.18	0.26	15
4	0.03	0.05	0.20	0.32	0.29	20
5	0.10	0.16	0.22	0.21	0.32	25
6	0.18	0.22	0.28	0.28	0.31	30
7	0.05	0.05	0.32	0.165	0.23	35
8	0.25	0.33	0.32	0.32	0.33	40
9	0.11	0.25	0.15	0.29	0.33	45
10	0.07	0.09	0.21	0.23	0.29	50

Table 7.5: Packet error rates and $E[H_1]$ values for 10-user system, $B = 0.5$.

Finally, the performance comparison is shown in Figures 7.8a and 7.8b (page 106). We see that CAOS outperformed other policies. NCQRS tracked very near CAOS, while the performance of DBP was more than three times worse, compared to NCQRS.

7.4.2 $\overline{V}_T(E[H_1])$ with Fixed Burstiness

In this experiment, we compared the $\overline{V}_T$ of policies between setups that had different bad-state holding times. For a 10 user system with $B = 0.7$, we drew

$E[H_1]$ values from the set $\{2, 8, 15, 22, 29, 36, 43, 50\}$ (setting them identically for all streams at each time). We repeated the above scenario three times, once for $\overline{p} = 0.1$, then for 0.2 and finally, for $\overline{p} = 0.3$. We discovered that the examined functions $\overline{V}_T(E[H_1])$ are similar for all three levels of PER. Therefore, in Figure 7.9 on page 107 we outline only the case of $\overline{p} = 0.2$.

We learned from this experiment that the CAOS heuristic seems to be dominant throughout a range of bad-state holding times. Nevertheless, DBP scheduling showed violation of around 7-8% of the time (on average), while the NCQRS did introduce a violation around 1%. Moreover, the examined range of state-holding times did not seem to upset the performance of any examined policy as it went up. One may consider to examine a wider range of $E[H_1]$ values as future work.

7.4.3 $\overline{V}_T(E[H_1], B)$ in PER-Homogeneous Scenarios

Here we again compared the performance of our policies in terms of $\overline{V}_T$. In this case we experimented with systems of 5 and 15 substations. The channels were fully homogeneous. The parameters such as PER, $E[H_1]$ and B were identical for every user. We looked at $\overline{V}_T$ as function of $E[H_1]$ at various degrees of burstiness but in particular, while $B \in \{0.3, 0.5, 0.7, 0.9\}$.

Figure 7.10 and Figure 7.11 (page 107) for 5 and 15 users demonstrate a very similar response. Additionally, for $B = 0.5$, the performance of CAOS and NCQRS is better compared to the case of $B = 0.3$. On other hand, the DBP was barely affected by changes in B. Additionally, for $B = 0.5$, the CAOS policy managed to almost avoid violation (on average). NCQRS kept the $\overline{V}_T$ below 2%. DBP performed within approximately 8%, which is still

much worse than the other policies.

We found that the average performance was even better in the 15-user system for CAOS and NCQRS, compared to the 5-user setup. That could be explained by the availability of a larger amount of retransmit slots – 30 slots for 15 users, compared to 10 slots in 5-user setup. In the examined parameter space, such a setting would allow some streams to better cope with bursts. The probability that others experience a similar condition during the same time is low.

Figure 7.12 summarizes the measurements for various degrees of burstiness. It shows the NCQRS policy function $\overline{V}_T\left(E[H_1]\right)$ for four levels of channel burstiness: $0.3, 0.5, 0.7$ and 0.9, given $\overline{p} = 0.2$. We observed the performance to be generally worse for smaller values of B. This made sense, since for smaller values of B, the channel is expected to be disrupted for longer. This also corresponds to results of Willig (2005) for Gilbert-Elliot channels with respect to a change in burstiness.

7.4.4 $\overline{V}_T(\overline{p})$ in Fully Heterogeneous Scenarios

We investigated the performance of 5-user and 15-user systems over a setup in which each link was assigned with a unique value of $p, E[H_1]$ and B. Table 7.6 shows all values for the 5-user system, while the data for the 15-user system is presented in Table 7.7.

The plots for performance comparison are presented in Figures 7.13a and 7.13b (page 110) for the 5-user and 15-user systems respectively. In both scenarios, we observed that our learning policy performs better than DBP, but worse than CAOS. The latter managed to beat the other policies in every

Nodes	$\overline{p} = 0.1$	$\overline{p}_e = 0.15$	$\overline{p} = 0.2$	$\overline{p} = 0.25$	$\overline{p} = 0.3$	$E[H_1]$	B
1	0.2	0.2	0.3	0.3	0.35	5	0.74
2	0.15	0.23	0.28	0.25	0.22	7	0.45
3	0.09	0.15	0.17	0.19	0.36	15	0.66
4	0.04	0.04	0.08	0.13	0.15	20	0.46
5	0.02	0.13	0.17	0.38	0.42	25	0.88

Table 7.6: G.E. Channel model values for 5-user system in a fully heterogeneous scenario.

Nodes	$\overline{p} = 0.1$	$\overline{p} = 0.15$	$\overline{p} = 0.2$	$\overline{p} = 0.25$	$\overline{p} = 0.3$	$E[H_1]$	B
1	0.2	0.2	0.3	0.3	0.33	5	0.5
2	0.15	0.25	0.28	0.2	0.2	7	0.45
3	0.1	0.15	0.17	0.17	0.32	15	0.51
4	0.05	0.05	0.05	0.05	0.15	20	0.46
5	0.03	0.15	0.15	0.33	0.4	25	0.88
6	0.15	0.2	0.18	0.22	0.25	21	0.4
7	0.2	0.21	0.25	0.32	0.35	17	0.77
8	0.04	0.13	0.08	0.35	0.37	30	0.68
9	0.04	0.05	0.27	0.29	0.35	40	0.89
10	0.05	0.13	0.19	0.26	0.23	45	0.38
11	0.1	0.2	0.19	0.3	0.38	2	0.7
12	0.2	0.25	0.32	0.32	0.32	12	0.9
13	0.05	0.12	0.25	0.26	0.32	15	0.5
14	0.03	0.03	0.17	0.23	0.28	8	0.45
15	0.11	0.13	0.15	0.15	0.25	2	0.51

Table 7.7: G.E. Channel model values for 15-user system in a fully heterogeneous scenario.

Markov channel experiment that we performed in this work.

7.5 Convergence

We studied the convergence of our learning scheduling policy. The method that was developed to determine its convergence speed is explained in Section

6.3.3, page 83.

For the nonoverlapping batch-means, we used batches of 800 super frames ($L = 800$). The duration of a single replication run was 10^6 slots. Step-changes in PER were always generated at $t_s = 150,000$ $[slots]$. Additionally, we set the threshold ω (see Equation 6.7) to 80% ($\omega = 0.8$). This parameter is used to "balance" the comparison between batch averages and the known steady-state mean.

7.5.1 Convergence as Function of Step Size

In this part we began with an initial step, $p_s = 0.1$. For all streams, we measured the convergence time using Equations 6.7 and 6.8 and assigned different values for p_e each time. We performed five independent trials: $p_h \in \{0.2, 0.25, 0.3, 0.35, 0.4\}$. For each new value of p_h, the convergence time was evaluated. We averaged each trial using values of $n = 150$ replications. We defined all streams with $(5, 10)$-firm deadlines in this experiment. We repeated this experiment for systems of 10, 15 and 20 substations. The convergence times are shown in Figure 7.14 (page 111).

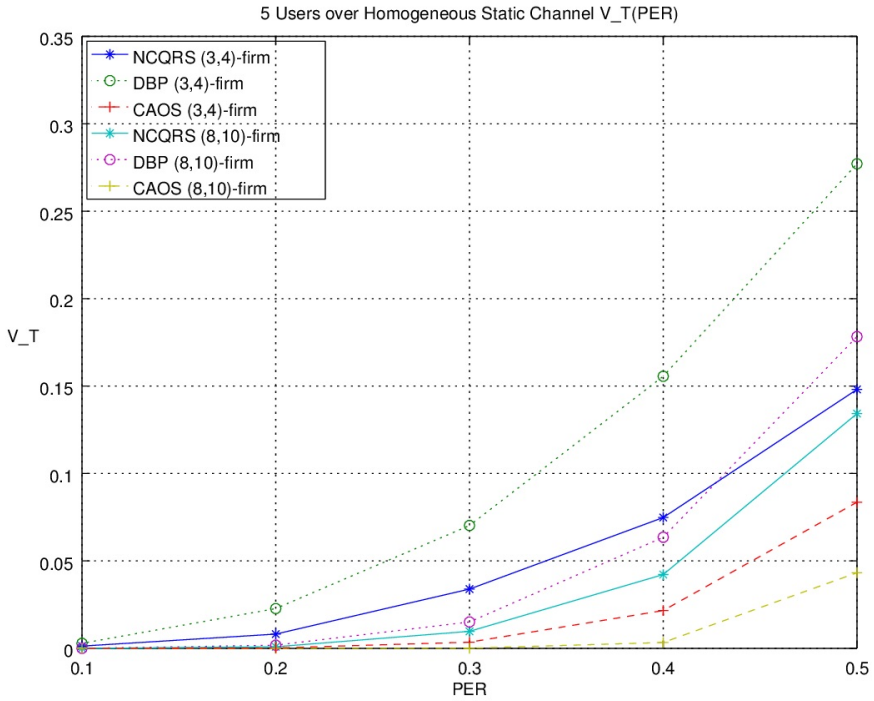
As a result, we could not find any visible dependency of convergence time with respect to the size of change in packet error rate. It may well happen that the examined factor has no effect on policy convergence speed. In addition, the measured convergence time varied between 5000 and 8000 super frames[5]. The change in PER was an ideal step, applied to all substations at once. It could be therefore interpreted as the most extreme change in PER for a wireless environment.

[5] If we assume one slot is 1 ms, then convergence time varies between 100-160 seconds.

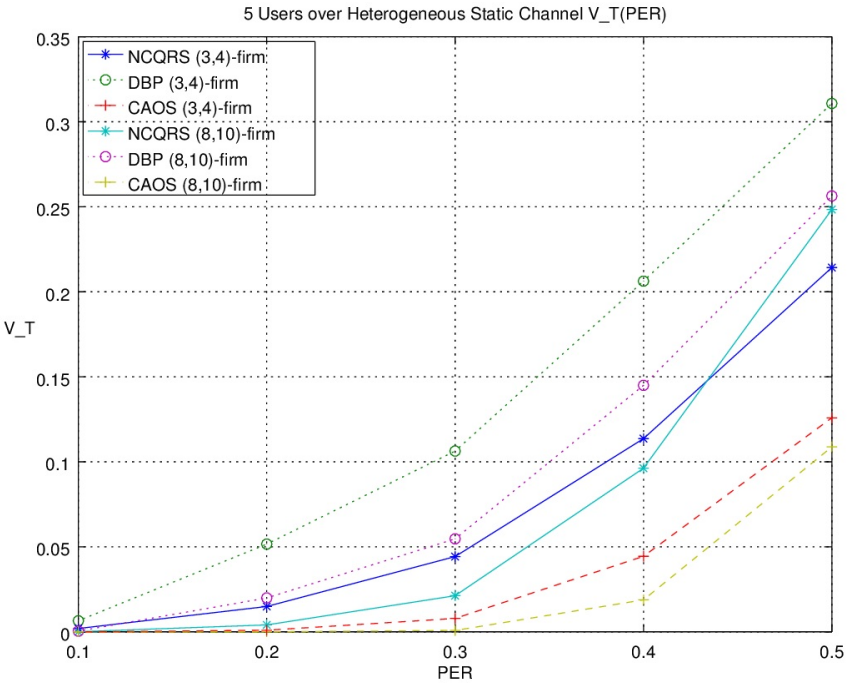
7.5.2 Convergence as Function of (m, k)-Firm Deadlines

For this convergence experiment, we used fixed step parameters $p_s = 0.2, p_e = 0.4$ and tried different (m, k)-firm settings. We then estimated the system's convergence time for each one of the firm deadline settings. Arbitrary m's and k's were selected with a condition, that the algorithm is able to converge. This experiment was repeated for 10, 15 and 20 substations scenarios, using the following pairs: $(4, 6)$, $(6, 8)$, $(8, 10)$, $(10, 12)$, $(12, 14)$ and $(14, 16)$-firm.

Figure 7.15 (page 112) demonstrates a slight increase in convergence times, as function of the growing m's and k's. This is an expected outcome, because we had previously learned that due to an increase in k, the RL algorithm would need more time to visit all its states, hence, it will take it longer to converge to new values.

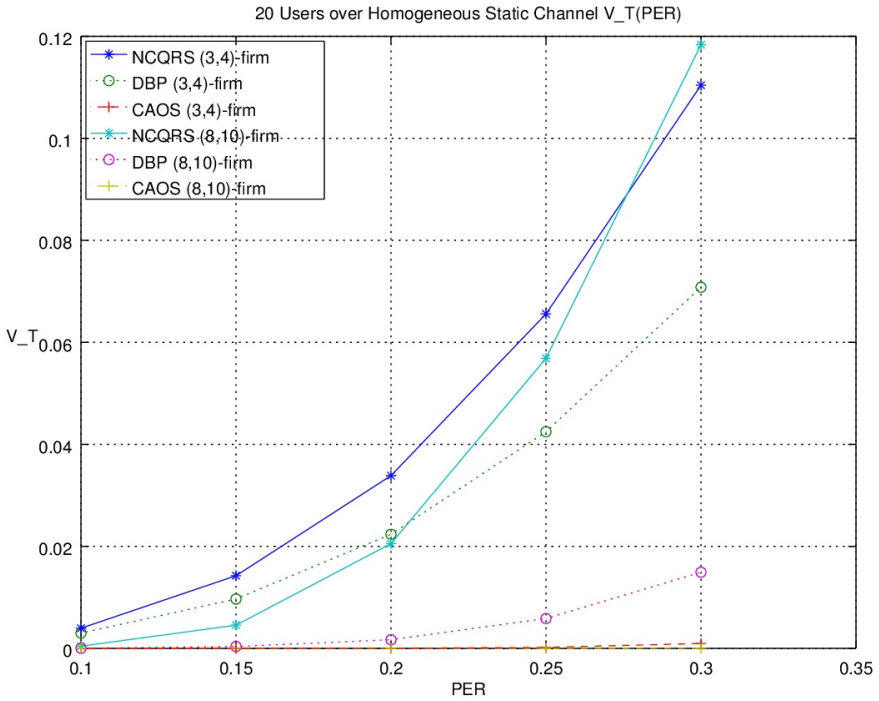


(a) p_{hom} static channel.

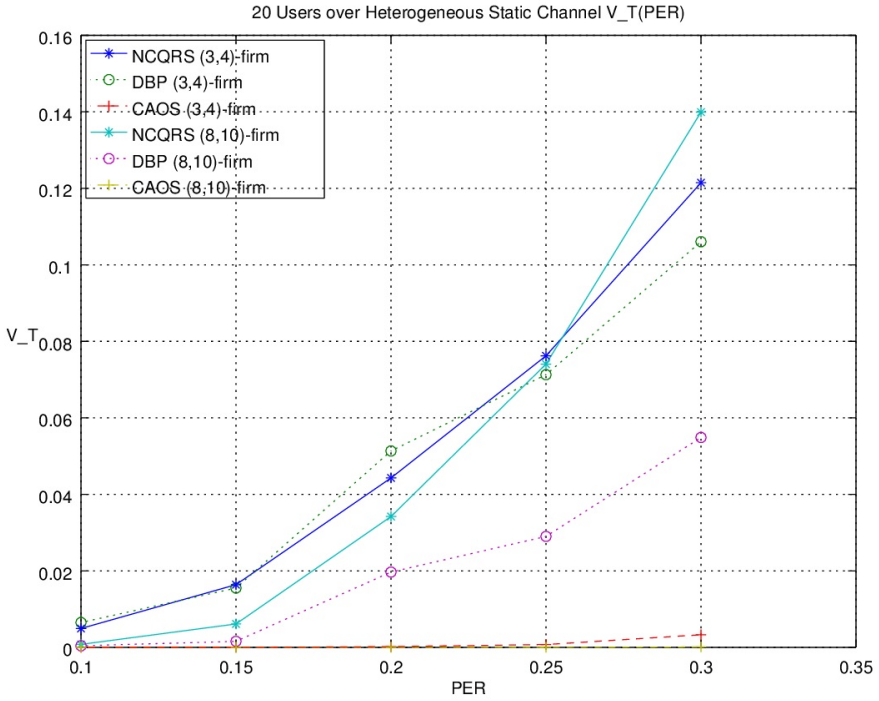


(b) p_{het} static channel.

Figure 7.6: $\overline{V}_T(\overline{p})$ over Static channel, 5 users.

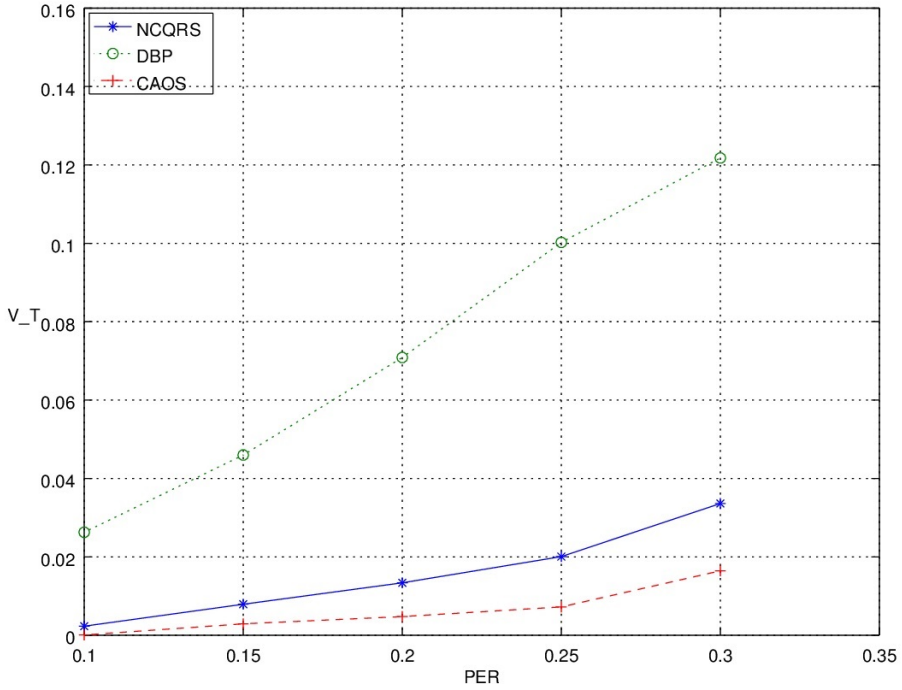


(a) p_{hom} static channel.

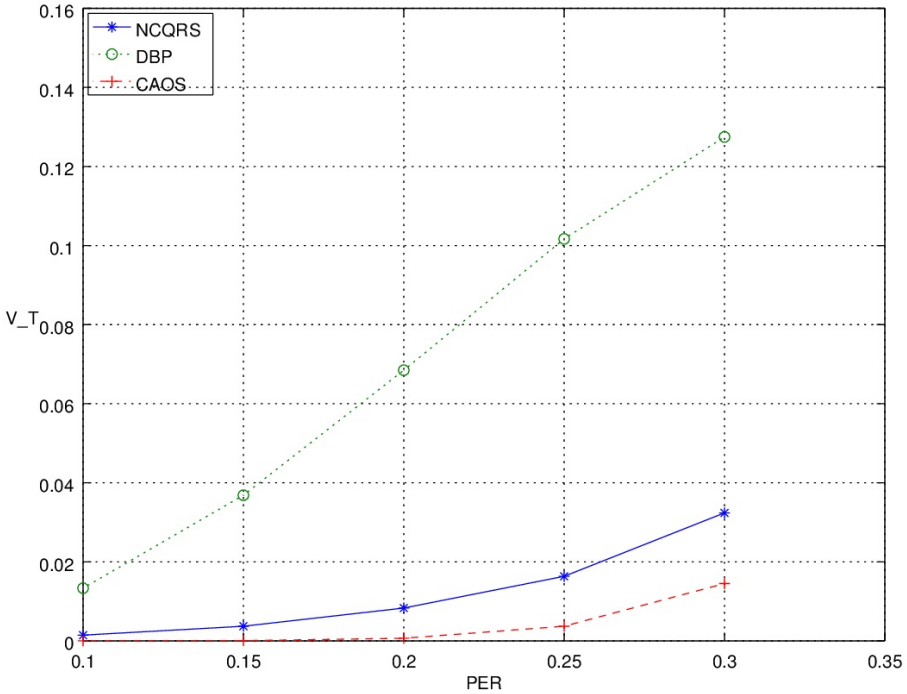


(b) p_{het} static channel.

Figure 7.7: $\overline{V}_T(\overline{p})$ over Static channels, 20 users.



(a) 10 users, B=0.5, heterogeneous error rates and bad holding times



(b) 20 users, B=0.5, homogeneous error rates and heterogeneous bad holding times

Figure 7.8: $\overline{V}_T(\overline{p})$ over G.E. channels, 10 and 20 users.

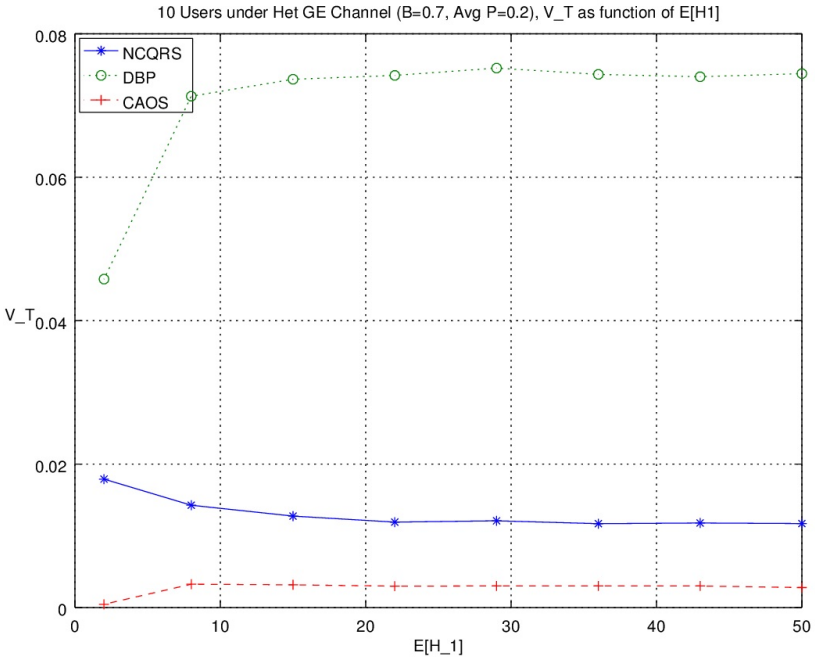


Figure 7.9: $\overline{V}_T\left(E[H_1]\right)$ over G.E. heterogeneous setups, $\overline{p} = 0.2$.

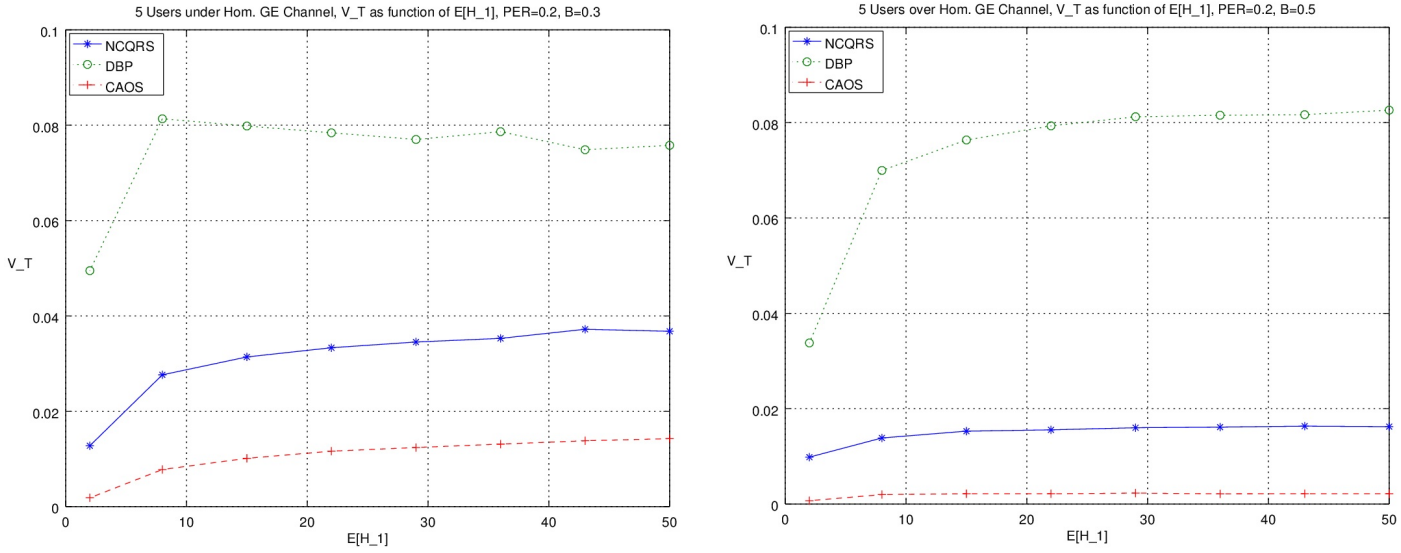


Figure 7.10: $\overline{V}_T\left(E[H_1]\right)$ in 5-user system over G.E. homogeneous setup, $\overline{p} = 0.2$, $B = 0.3$ (left) and $B = 0.5$ (right).

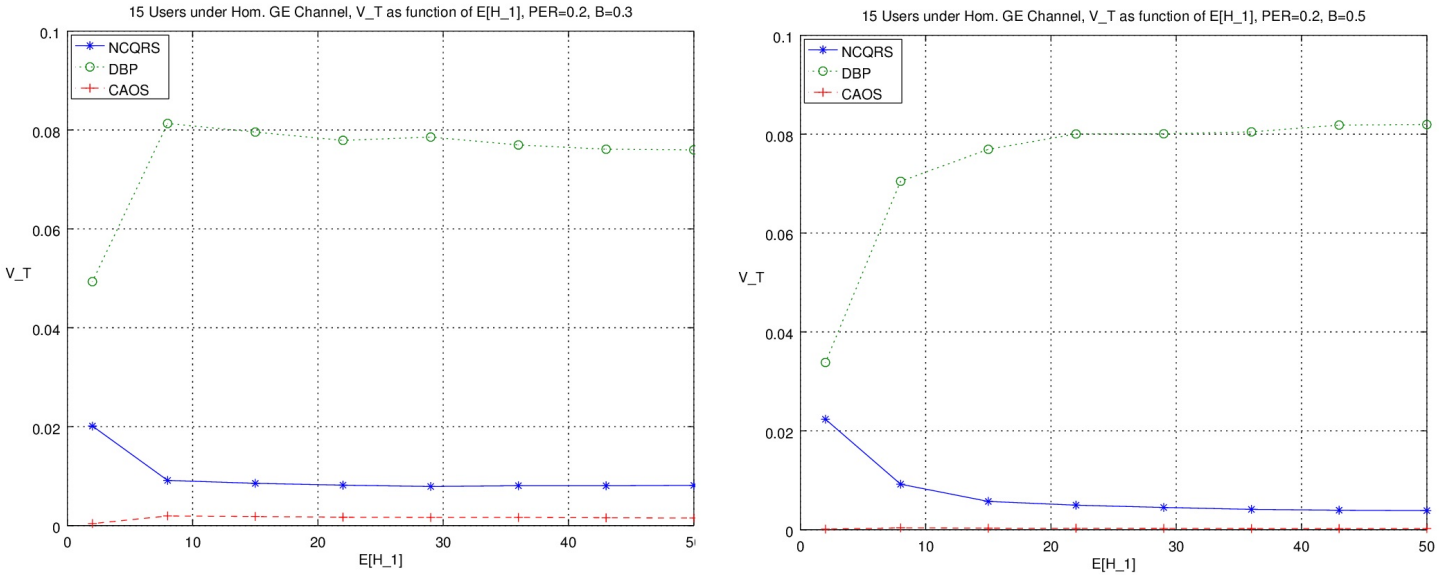


Figure 7.11: $\overline{V}_T\left(E[H_1]\right)$ in 15 user system over G.E. homogeneous setup, $\overline{p} = 0.2$, $B = 0.3$ (left) and $B = 0.5$ (right).

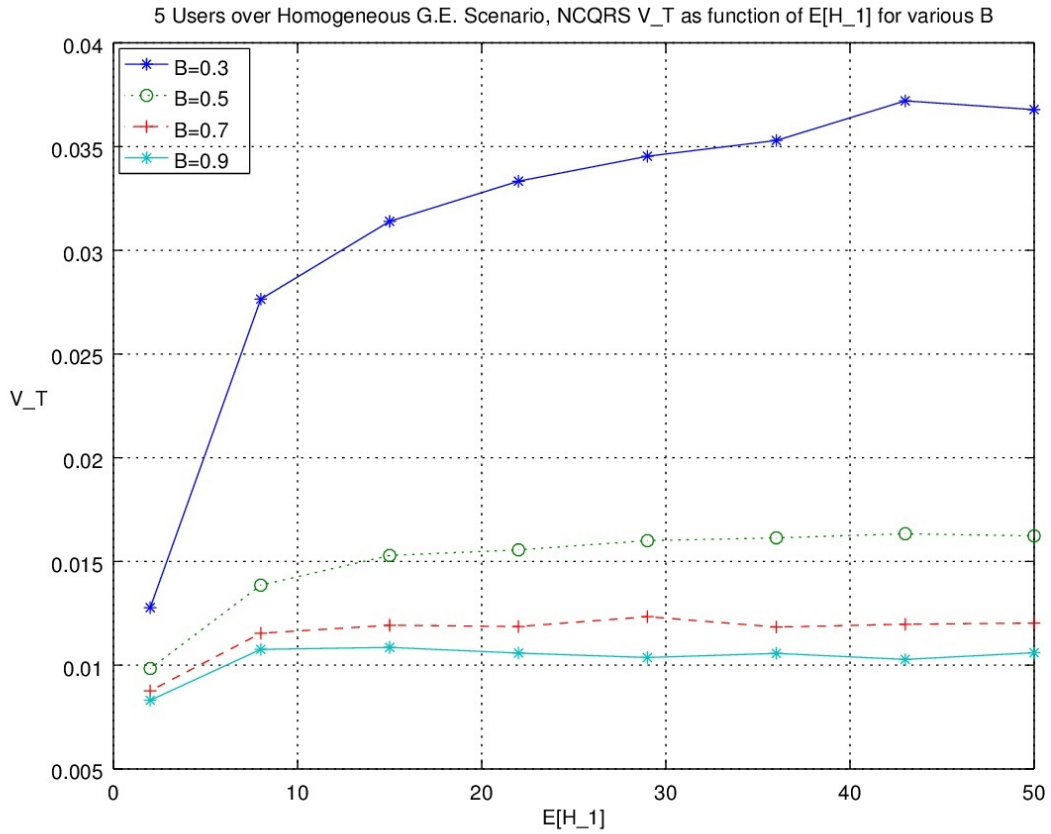
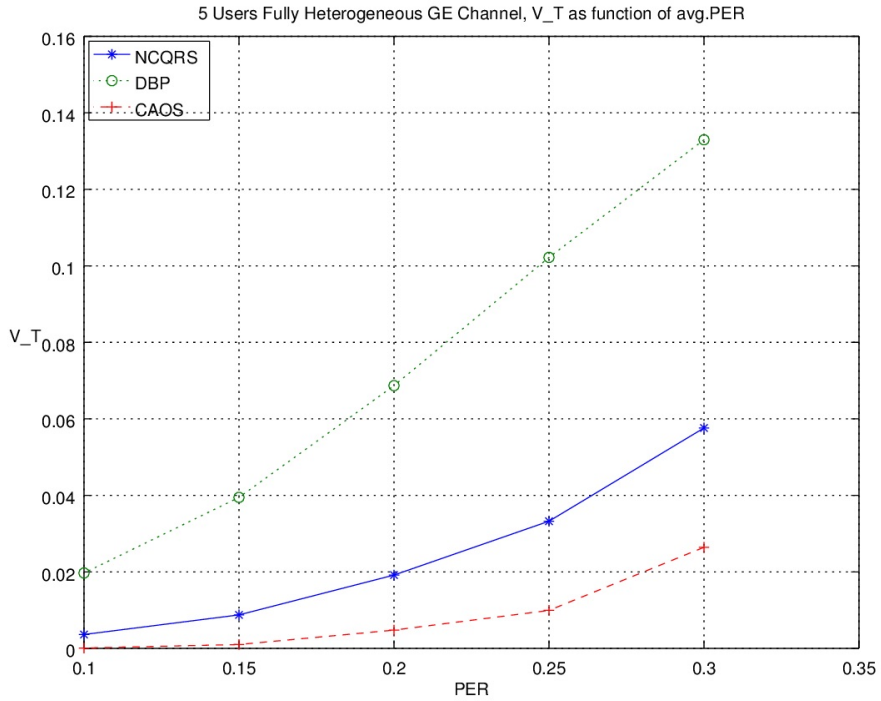
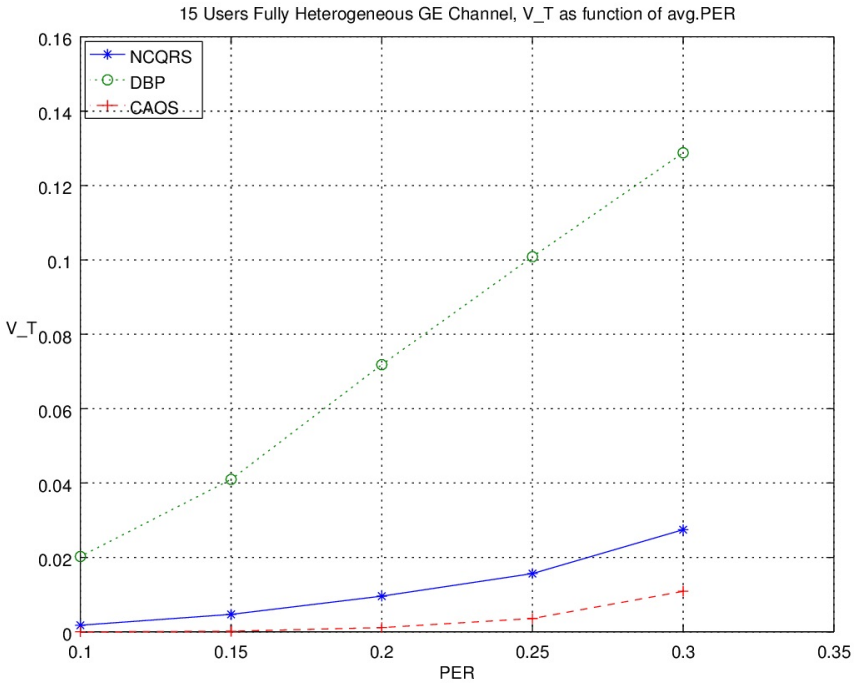


Figure 7.12: $\overline{V}_T\left(E[H_1]\right)$ in G.E. p_{hom} scenarios for $\overline{p} = 0.2$, 5 users, few values of B.



(a) 5 users



(b) 15 users

Figure 7.13: $\overline{V}_T(\overline{p})$ over fully heterogeneous G.E. setups, 5 and 15 users.

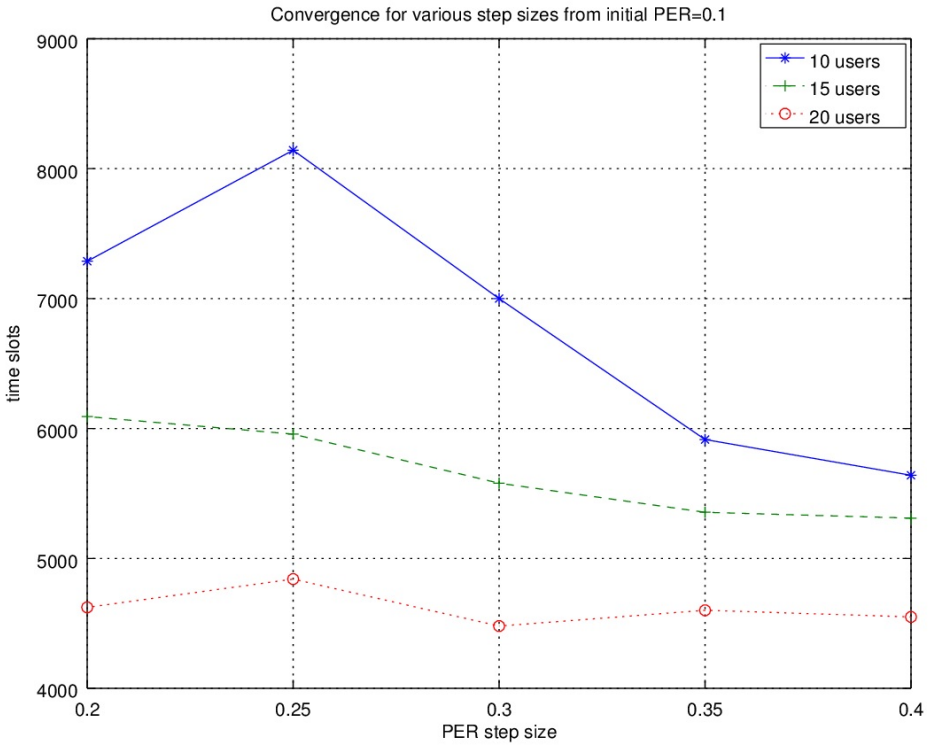


Figure 7.14: Convergence time (in super frames) as function of p_h for 10,15 and 20 users, over Static homogeneous channels setup, $p_s = 0.1, L = 800, \omega = 0.8$.

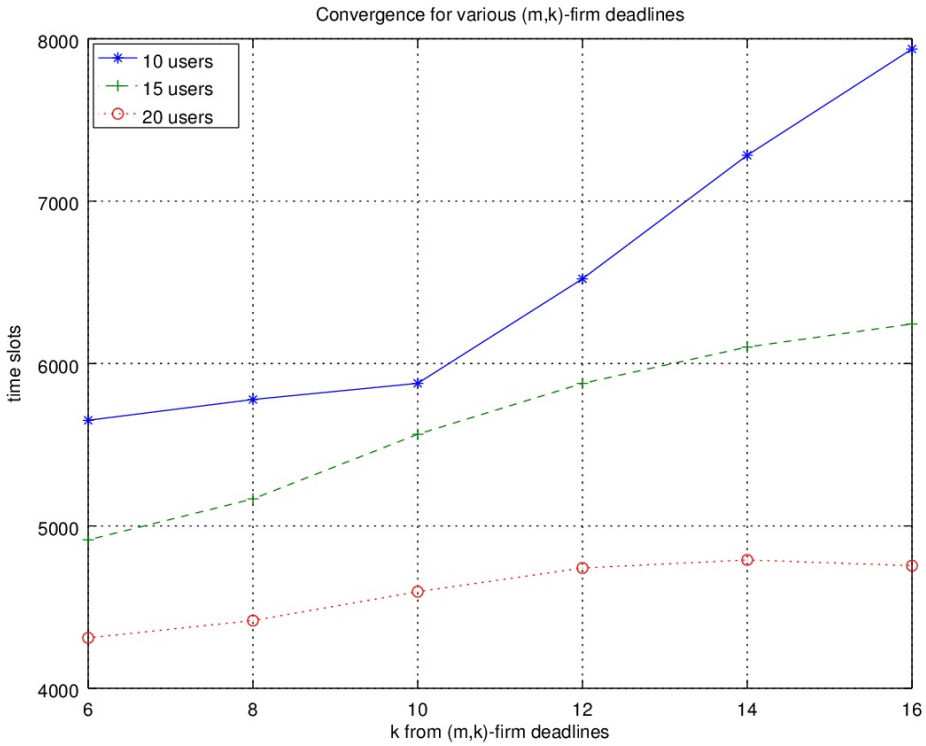


Figure 7.15: Convergence time (in superframes) as function of (m, k) for 10, 15 and 20 users over Static homogeneous channels setup, $p_s = 0.2, p_e = 0.4$.

Chapter VIII

Conclusions

The results of this thesis represent a step in the investigation of new scheduling policies that serve many streams defined with (m, k)-firm deadline constraints. We compared the performance of several policies over wireless links and in particular, over channels with different channel qualities and time-varying characteristics. We also observed how various channels properties may affect efficiency of link-retransmissions scheduling.

The main finding was that our reinforcement learning policy failed to outperform the CAOS policy but performed better than the DBP policy. In some cases the performance presented by NCQRS heuristic, was very close, sometimes equal or better than CAOS.

Both the NCQRS and DBP policies store information about the past states of input streams in order to make decisions about their next super frame retransmissions. Although the CAOS policy ignores such history and cares only about estimated packet error rates it outperforms the other two.

Based on this work's completed experimentation, we consider CAOS as the best candidate to reduce the long-term average violation rate. The complexity of this algorithm is low compared to NCQRS, in terms of both memory and computation. However, the assumptions that were made upon the structure of the reward function, rewards coefficients and upon other elements of learning may have introduced a suboptimal performance, hence, might potentially be improved in the future.

References

3GPP. (2008, September). *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification* (TS No. 36.322). 3rd Generation Partnership Project (3GPP). Retrieved from `http://www.3gpp.org/ftp/Specs/html-info/36322.htm`

Abeni, L., & Buttazzo, G. (1999). Qos guarantee using probabilistic deadlines. In *Real-time systems, 1999. proceedings of the 11th euromicro conference on* (p. 242-249).

Anwar, M., & Xia, Y. (2014). Ieee 802.15.4e lldn: Superframe configuration for networked control systems. *33rd Chinese Control Conference (CCC)*, 5568-5573.

Bab, A., & Brafman, R. I. (2008). Multi-agent reinforcement learning in common interest and fixed sum stochastic games: An experimental study. *Journal of Machine Learning Research 9*, 2635-2675.

Bentley, J., & Yao, A. C.-C. (1976, August). An almost optimal algorithm for unbounded search. *Information Processing Letters*, *5*, 82-87.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming.* Belmont, Massachusetts: Athena Scientific.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on theoretical aspects of rationality and knowledge* (pp. 195–210). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bowling, M. (2005). Convergence and no-regret in multiagent learning. In *In advances in neural information processing systems 17* (pp. 209–216). MIT Press.

Bowling, M., & Veloso, M. (2000). *An analysis of stochastic game theory for multiagent reinforcement learning* (Tech. Rep. No. CMU-CS-00-165). Computer Science Department, Carnegie Mellon University.

Bowling, M., & Veloso, M. (2001). Rational and convergent learning in stochastic games. In *Proceedings of the 17th international joint con-*

ference on artificial intelligence - volume 2 (pp. 1021–1026). Morgan Kaufmann Publishers Inc.

Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. New York, NY, USA: Cambridge University Press.

Busoniu, L., Babushka, R., & Schutter, B. D. (2010). Multi-agent reinforcement learning: An overview. *Studies in Computational Intelligence*, *310*, 183-221.

Chang, C.-Y., & Hsiao, H.-F. (2015, July). Priority-base retransmission scheduling for automatic repeat request over wireless networks. In *Ubiquitous and future networks (icufn), 2015 seventh international conference on* (p. 350-354). doi: 10.1109/ICUFN.2015.7182563

Chen, D., Nixon, M., Han, S., Mok, A., & Zhu, X. (2014, Feb). Wirelesshart and ieee 802.15.4e. In *Industrial technology (icit), 2014 ieee international conference on* (p. 760-765).

Chen, F., German, R., & Dressler, F. (2010, March). Towards IEEE 802.15.4e: A Study of Performance Aspects. In *Proc. 8th ieee international conference on pervasive computing and communications (percom workshops)* (p. 68-73). Mannheim, Germany.

Chen, J., Song, Y., Wang, Z., & Sun, Y. (2004). Equivalent matrix dbp for streams with (m,k)-firm deadline. *2004 IEEE International Symposium on Industrial Electronics*, *1*, 675–680.

Chua, T., & Pheanis, D. (2006, Nov). Qos evaluation of sender-based loss-recovery techniques for voip. *Network, IEEE*, *20*(6), 14-22.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI '98/IAAI '98 Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, 746-752.

Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to algorithms* (2nd ed.). McGraw-Hill Higher Education.

Decotignie, J., & Pleinevaux, P. (1993). A survey on Industrial Communication Networks. *Annales Des Télécommunications*, *48*, 435–448.

Demarch, D., & Becker, L. (2007, July). An integrated scheduling and re-

transmission proposal for firm real-time traffic in ieee 802.11e. In *Real-time systems, 2007. ecrts '07. 19th euromicro conference on* (p. 146-158). doi: 10.1109/ECRTS.2007.83

Elliot, E. O. (1963). Estimate of error rates for codes on burst-noise channels. *Bell Systems Technical Journal*, *42*, 1977 - 1997.

Gamba, G., Tramarin, F., & Willig, A. (2009, Sept). Retransmission strategies for cyclic polling over wireless channels in the presence of interference. In *Emerging technologies factory automation, 2009. etfa 2009. ieee conference on* (p. 1-8). doi: 10.1109/ETFA.2009.5347042

Garcia, A. L., & Widjaja, I. (2004). *Communication Networks. Fundamental Concepts and Key Architectures*. McGraw-Hill.

Gibson, J. D. (Ed.). (2001). *Multimedia communications: Directions and innovations*. Orlando, FL, USA: Academic Press, Inc.

Gilbert, E. N. (1960). Capacity of a burst noise channel. *Bell Systems Technical Journal*, *39*, 1253-1265.

Gungor, V. C., & Hancke, G. P. (2009, October). Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Transactions on Industrial Electronics*, *56*(10), 4258-4265.

Hamdaoui, M., & Ramanathan, P. (1995). Dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, *44*, 1443-1451.

Hamdaoui, M., & Ramanathan, P. (1997). Evaluating dynamic failure probability for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, *46*, 1325-1337.

Hester, T., Lopes, M., & Stone, P. (2013, May). Learning exploration strategies in model-based reinforcement learning. In *The twelfth international conference on autonomous agents and multiagent systems (aamas).*

Huang, J., Yang, B., & Liu, D. (2005, Aug). A distributed q-learning algorithm for multi-agent team coordination. In *Machine learning and cybernetics, 2005. proceedings of 2005 international conference on* (Vol. 1, p. 108-113).

Hunter, J. S. (1986, Oct). The exponentially weighted moving average. *Journal of Quality technology*, *18*(4), 203-210.

IEEE Standard for LR-WPANs. (2012, April). Ieee standard for local and

metropolitan area networks – part 15.4: Low-rate wireless personal area networks (lr-wpans) – amendment 1: Mac sublayer [Computer software manual]. (IEEE Std 802.15.4e-2012)

Jain, R. K. (1991). *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling.* ACM.

Kapetanakis, S., & Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. *In proceeding 18-th national conference on Artificial intelligence*, 326–331.

Kim, K.-I. (2010). A novel scheduling for (m, k)-firm streams in wireless sensor networks. *Networked Computing and Advanced Information Management*, 553-556.

Kim, K.-I. (2011). A revised speed protocol for (m,k)-firm streams in wireless sensor networks. *International Conference on ICT Convergence*, 267–268.

Kim, K.-I., & Li, B. (2012). A congestion control scheme for (m, k)-firm realtime streams in wireless sensor networks. *International Conference on ICT Convergence*, 593-594.

Kim, S., Fonseca, R., & Culler, D. (2004). Reliable transfer on wireless sensor networks. *IEEE Communication Society Conference on Sensor and Ad Hoc Communications and Networks*, 449–459.

Knuth, D. E. (1998). *The art of computer programming, volume 3: (2nd ed.) sorting and searching.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

Kurose, J. F., & Ross, K. (2002). *Computer networking: A top-down approach featuring the internet* (2nd ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Law, A., & Kelton, W. (2000). *Simulation modeling and analysis, 3rd ed.* McGraw Hill.

Li, B., & Kim, K.-I. (2012). A novel routing protocol for (m, k)-firm-based real-time streams in wireless sensor networks. *Wireless Communications and Networking Conference*, 1715-1719.

Li, B., & Kim, K.-I. (2013). A real-time routing protocol for (m,k)-firm streams in wireless sensor networks. *IEEE Intelligent Sensors, Sensor*

Networks and Information Processing, 129–134.

Lin, S., Costello, J., D.J., & Miller, M. (1984, December). Automatic-repeat-request error-control schemes. *Communications Magazine, IEEE*, *22*(12), 5-17.

Lindsay, W., & Ramanathan, P. (1997). Dbp-m: a technique for meeting end-to-end (m, k)-firm guarantee requirement in point-to-point networks. *Proceedings., 22nd Annual Conference on Local Computer Networks*, 294–303.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *In Proc. 11th International Conference on Machine Learning*, 157-163.

Littman, M. L. (2001a). Friend-or-foe q-learning in general-sum games. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 322-328.

Littman, M. L. (2001b). Value-function reinforcement learning in markov games. *Journal of Cognitive Systems Research 2*, 55-66.

Lopes, M., Lang, T., Berlin, F., Toussaint, M., Berlin, F., & yves Oudeyer, P. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *In neural information processing systems (nips).*

Lu, M., Steenkiste, P., & Chen, T. (2007, February). A time-based adaptive retry strategy for video streaming in 802.11 wlans: Research articles. *Wireless Communication Mobile Computing*, *7*(2), 187–203.

Maheswari, K., & Punithavalli, M. (2009, Dec). Receiver based packet loss replacement technique for high quality voip streams. In *Nature biologically inspired computing, 2009. nabic 2009. world congress on* (p. 1669-1672).

Mahmood, M., Seah, W., & Welch, I. (2015). Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, 166-187.

Matsumoto, M., & Nishimura, T. (1998, January). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, *8*(1), 3–30.

Medidi, S., Nandanavanam, V., & Medidi, M. (2011, Oct). Reliable sensor-

to-sink data transfer with duty cycles for wireless sensor networks. In *Local computer networks (lcn), 2011 ieee 36th conference on* (p. 358-365).

Nonnenmacher, J., Biersack, E., & Towsley, D. (1998, Aug). Parity-based loss recovery for reliable multicast transmission. *Networking, IEEE/ACM Transactions on*, *6*(4), 349-361.

Norris, J. R. (1997). *Markov chains*. Cambridge, UK: Cambridge University Press.

Pawlikowski, K. (1990, June). Steady-state simulation of queueing processes: Survey of problems and solutions. *ACM Comput. Surv.*, *22*(2), 123–170.

Perkins, C., Hodson, O., & Hardman, V. (1998). A survey of packet loss recovery techniques for streaming audio. *Network*, *12*, 40–48.

Poggi, E., Song, Y., Koubaa, A., & Wang, Z. (2003). Matrix-dbp for (m, k)-firm real-time guarantee.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming* (1st ed.). New York, NY, USA: John Wiley & Sons, Inc.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257 - 286.

Radmand, P., Talevski, A., Petersen, S., & Carlsen, S. (2010, April). Comparison of industrial wsn standards. In *4th ieee international conference on digital ecosystems and technologies* (p. 632-637). doi: 10.1109/DEST.2010.5610582

Ramabhadran, S., & Pasquale, J. (2006, Dec). The stratified round robin scheduler: Design, analysis and implementation. *Networking, IEEE/ACM Transactions on*, *14*(6), 1362-1373. doi: 10.1109/TNET.2006.886287

Ramanathan, P. (1999). Overload management in real-time control applications using (m, k)-firm guarantee. *Parallel and Distributed Systems, IEEE Transactions on*, *10*(6), 549-559.

Robbins, H., & Monro, S. (1951, 09). A stochastic approximation method. *Ann. Math. Statist.*, *22*(3), 400–407.

Saha, D., Mukherjee, S., & Tripathi, S. (1998, Dec). Carry-over round robin: a simple cell scheduling mechanism for atm networks. *Networking, IEEE/ACM Transactions on*, *6*(6), 779-796. doi: 10.1109/90.748089

Schaerf, A., Shoham, Y., & Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, *2*, 475–500.

Semprebom, T., Montez, C., Moraes, R., & Vasques, F. e. (2009). Distributed dbp: A (m,k)-firm based distributed approach for qos provision in ieee 802.15.4 networks. *IEEE Emerging Technologies and Factory Automation*, 1-8.

Shaikh, M., & Ahmed, K. (2009). Investigation on data loss of cbr video streams over wireless channels. *6th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 58–65.

Shapley, L. (1953). Stochastic games. In *Stochastic games* (Vol. 39, pp. 1095–1100). Princeton University.

Shoham, Y., & Tennenholtz, M. (1992). On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the tenth national conference on artificial intelligence* (pp. 276–281). AAAI Press.

Singh, S., Jaakkola, T., Littman, M., & Szepesvári, C. (1998). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 287-308.

Song, Y., Koubaa, A., & Simonot, F. (2002). Switched ethernet for real-time industrial communication: modelling and message buffering delay evaluation. In *Factory communication systems, 2002. 4th ieee international workshop on* (p. 27-35). doi: 10.1109/WFCS.2002.1159697

Stankovic, J., Abdelzaher, T., Lu, C., Sha, L., & Hou, J. (2003, July). Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, *91*(7), 1002-1022. doi: 10.1109/JPROC.2003.814620

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (1st ed.). Cambridge, MA, USA: MIT Press.

Sze, H., Liew, S., & Lee, Y. (2001, March). A packet-loss-recovery scheme for continuous-media streaming over the internet. *Communications Letters, IEEE*, *5*(3), 116-118.

Tasaka, S. (1986). *Performance analysis of multiple access protocols.* Cambridge, MA, USA: MIT Press.

Tokic, M. (2010). Adaptive e-greedy exploration in reinforcement learning based on value differences. In *Proceedings of the 33rd annual german conference on advances in artificial intelligence* (pp. 203–210). Springer-Verlag.

Tovar, E., & Vasques, F. (1999). Real-time fieldbus communications using Profibus networks . *IEEE Transactions on Industrial Electronics*, 1241–1251.

Tse, D., & Viswanath, P. (2005). *Fundamentals of wireless communication.* New York, NY, USA: Cambridge University Press.

Varga, A. (2001). The omnet++ discrete event simulation system. In *Esm'01.*

Vitturi, S., Tramarin, F., & Seno, L. (2013, June). Industrial Wireless Networks: The Significance of Timeliness in Communication Systems. *IEEE Industrial Electronics Magazine*, *7*(2), 40-51.

Wang, X., & Sandholm, T. (2002). Reinforcement learning to play an optimal nash equilibrium in team markov games. In *in advances in neural information processing systems* (pp. 1571–1578). MIT Press.

Wang, Z., qiong Song, Y., Poggi, E.-M., & Sun, Y. (2002). A survey of weakly-hard real time schedule theory and its application. *In Proc. International Symposium on Distributed Computing and Applications to Business. Engineering and Science (DCABES).*

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* (Unpublished doctoral dissertation). King's College, Cambridge, UK.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279-292.

Willig, A. (2003, Aug). Polling-based mac protocols for improving real-time performance in a wireless profibus. *IEEE Transactions on Industrial Electronics*, *50*(4), 806-817. doi: 10.1109/TIE.2003.814992

Willig, A. (2005). Scheduling multiple streams with (m,k)-firm deadlines having different importance over markovian channels. *10th IEEE Conference on Emerging Technologies and Factory Automation*, *1*, 79-85.

Willig, A., Matheus, K., & Wolisz, A. (2005). Wireless Technology in Indus-

trial Networks. *Proceedings of the IEEE*, *93*, 1130–1151.